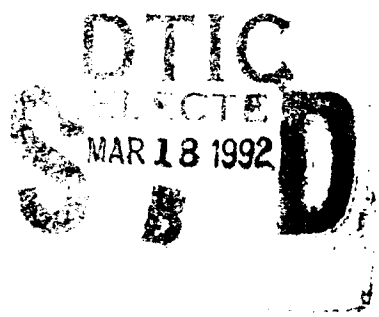
Technical Report 1463
November 1991

Extensibility Experiments with the Software Life-Cycle Support Environment

S. A. Parker
R. H. Mumm



NAVAL OCEAN SYSTEMS CENTER
San Diego, California 92152-5000

J. D. FONTANA, CAPT, USN
Commander

R. T. SHEARER, Acting
Technical Director

ADMINISTRATIVE INFORMATION

This report was done under the Computer Technology block program, Software Engineering for Command, Control, Communication Systems project, Software Engineering Environment Prototypes task. The work was done in FY 91 by S. A. Parker and R. H. Mumm of the Computer Software and Technology Branch, NOSC, Code 411.

Released by
G. Schulte, Head
Computer Software and
Technology Branch

Under authority of
A. G. Justice, Head
Information Processing
and Displaying Division

ACKNOWLEDGMENTS

The authors thank Marty Hogan and Tom Strelich, General Research Corporation and Dr. Michael Shapiro, Naval Ocean Systems Center, for reviewing this document and providing valuable comments. We really appreciate all the technical assistance from Marty Hogan.

JT

EXECUTIVE SUMMARY

OBJECTIVE

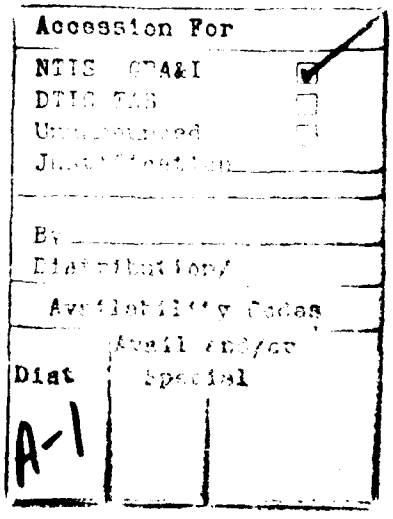
Our objective was to investigate the extensibility of the Software Life-Cycle Support Environment (SLCSE). The research focused on whether the environment could be tailored to meet the needs of specific Navy projects.

RESULTS

The SLCSE was successfully tailored to meet the needs of the Ship Gridlock project at the Naval Surface Weapons Center, Dahlgren. Eight tools and the ALS/N ADAVAX compiler were integrated into the environment.

RECOMMENDATIONS

1. The enhanced SLCSE must have a user interface that is significantly easier to use. An X Window implementation that is developed with assistance from human factors experts should help.
2. The enhanced SLCSE needs improved documentation. There should be a single user's guide for using the environment that includes a section on basic instructions for the novice user. There also needs to be a single user's guide for how to do tool integration. In the existing SLCSE tool integration instructions are incompletely described in a collection of manuals. All user instructions need to be beta tested. On-line user's guides would be helpful.
3. The enhanced SLCSE needs to provide an easier method for integrating tools. Users must be able to easily integrate their own tools.
4. We recommend that NOSC take an active role in the development of the enhanced SLCSE to ensure that the SLCSE provides the capabilities required by the Navy. This participation should include attending reviews and demonstrations as well as being a beta test site.



CONTENTS

1.0 INTRODUCTION 1

2.0 OVERVIEW OF THE SLCSE 3

2.1 BACKGROUND 3

2.2 MAJOR CHARACTERISTICS OF EXISTING SLCSE 4

2.2.1 Multiple User Roles 4

2.2.2 Common Database 5

2.2.3 Automated Document Generation 5

2.2.4 Common User Interface 5

2.2.5 Extensibility 6

2.2.6 Support for Multiple Languages 6

2.2.7 Support for Multiple Projects 7

2.2.8 SLCSE Toolset 7

2.3 COMMERCIAL SOFTWARE TO BE PURCHASED 10

2.4 HARDWARE REQUIREMENTS 10

2.5 CHARACTERISTICS OF THE FUTURE SLCSE 10

2.5.1 User Interface Enhancements 11

2.5.2 Repository Enhancements 11

2.5.3 Tool Enhancements 11

2.5.4 Tool Integration Enhancements 11

2.5.5 Hosting to Additional Platforms 12

2.5.6 Product Plans 12

3.0 SLCSE AS A PROJECT TOOL 13

3.1 SLCSE STRENGTHS 13

3.1.1 Tailoring of SLCSE Toolset 13

3.1.2 User Roles 13

3.1.3 Ease of Learning 13

3.1.4 Document Generation Capability 14

3.1.5 Stability 14

3.1.6 Tailoring of SLCSE Database 14

3.2 SOME SLCSE WEAKNESSES 14
3.2.1 User Interface 14
3.2.2 User's Manual 15
3.2.3 Error Messages 15
3.3 SUGGESTED ENHANCEMENTS TO SLCSE 15
4.0 SLCSE TOOL INTEGRATION 17
4.1 LEVELS OF TOOL INTEGRATION 17
4.1.1 Simple Tools 17
4.1.2 UI-Conformant Tools 17
4.1.3 Database Conformant Tools 18
4.2 HIGH-LEVEL STEPS FOR TOOL INTEGRATION 18
4.2.1 Integrating Simple Tools 20
4.2.2 Integrating UI-Conformant Tools 20
4.3 LESSONS LEARNED 21
5.0 INTEGRATING ALS/N TOOLS 25
5.1 INTEGRATION 25
5.2 ALS/N SETUP WINDOW DESCRIPTIONS AND USE 25
6.0 RECOMMENDATIONS 33
6.1 USER INTERFACE 33
6.2 EXTENSIBILITY 33
6.3 DOCUMENTATION 33
6.4 E-SLCSE AS NAVY C3 SOFTWARE DEVELOPMENT ENVIRONMENT 33
6.5 ADDITIONAL ON-LINE CAPABILITIES 34
7.0 REFERENCES 35
8.0 BIBLIOGRAPHY 36
APPENDICES
A SLCSE INSTRUCTIONS FOR THE NOVICE A-1
B DETAILED STEPS FOR TOOL INTEGRATION B-1
C SOURCE FILES FOR ALS/N WINDOWS C-1
D DESCRIPTION OF TOOLS ADDED TO SLCSE BY NOSC D-1

FIGURES

4-1 Tool integration process 19

4-2 BBoard setup window 22

5-1 ADAVAX setup window 26

5-2 LNKVAX setup window 28

5-3 EXPVMS setup window 29

5-4 IMPVAX setup window 30

1.0 INTRODUCTION

This report describes the research carried out on the Software Life-Cycle Support Environment (SLCSE*) by the Software Engineering Environment (SEE) prototypes task of the Software Engineering for C^3 Systems project. The focus of this investigation is to perform extensibility experiments with the SLCSE. These experiments included the development of an interface to the ALS/N ADAVAX compiler and the integration of a number of public domain and other no-cost software tools into the SLCSE. One goal of this research was to determine if the SLCSE could be tailored to meet the needs of a specific project.

The SLCSE was successfully tailored to meet the needs of the Ship Gridlock Project (SGP), Naval Surface Warfare Center (NSWC), Dahlgren. Most tools that the SGP is currently using, as well as others that would be useful to the project, were integrated into the SLCSE. The tools integrated include three developed by NSWC—a Language Generator Tool (LGEN), an electronic Bulletin Board (BBoard) program, and a Lexical Analyzer Generator Tool (LEXGEN); two developed by Naval Ocean Systems Center (NOSC), Code 411, Ada Primitive Compilation Order Tool (APRICOT) and Bit-Oriented Message Definer (BMD); and three from the Ada Software Repository (ASR) at White Sands—the NASA/Goddard Space Flight Center pretty printer, developed by AdaCraft; the Ada line counter, File_Checker, developed by Texas Instruments (T.I.), and the body stubber from the ASR, developed by Concurrent Computer Corporation.

NOSC Code 411 software engineers carried out this research with substantial assistance from General Research Corporation (GRC). GRC provided support in three ways—installation of the SLCSE, on-site classroom training on the SLCSE, and on-call consultation to answer SLCSE and tool integration questions. The research demonstrated environments that exist today, in this case the SLCSE, that can be tailored to meet the needs of specific projects.

This report includes the following

- Overview of the SLCSE
- Description of the future SLCSE
- Discussion of strengths/weaknesses of SLCSE
- Description of how to integrate tools
- Description of ALS/N integration
- Recommendations
- User Instructions for the Novice

* SLCSE is pronounced "slice."

- Detailed instructions for integrating tools
- Description of no-cost tools integrated

Potential users of this report are projects that use or plan to use the existing SLCSE. This report is beneficial to these projects even if they do not plan to tailor the SLCSE, but rather use it as is. The tool integration instructions provided are more detailed than any available from GRC. The instructions for the novice will also be useful to the new SLCSE user. This report provides helpful feedback to Rome Labs (RL) and to International Software Systems Incorporated (ISSI) for the development of the enhanced SLCSE. Furthermore, it may prove to be useful to the Software Technology for Adaptable Reliable Software (STARS) prime contractors. Tools integrated into the SLCSE under this project are useful for other Ada projects. All tools integrated under this project are available to DoD laboratories for free. The body stubber and Ada line counter were enhanced as part of this effort.

2.0 OVERVIEW OF THE SLCSE

2.1 BACKGROUND

The SLCSE is an Ada software development environment framework that was developed by GRC for RL as a proof-of-concept prototype. The SLCSE development is a continuation of work begun by the Software Technology for Adaptable Reliable Systems (STARS) Software Engineering Environment (SEE) team. The STARS SEE team developed an Operational Concept Document (OCD) (STARS JPO, 1985) containing a comprehensive set of requirements for a SEE to support the life-cycle development of software for the Department of Defense (DoD). The OCD provided a basis for the development of SLCSE. In 1984, the SLCSE Exploratory Development (6.2) research began with the definition of a SEE for life-cycle development of Air Force Command, Control, Communications, and Intelligence (C^3I) systems. Advanced Development (6.3) research began in 1986. This research was accomplished using an incremental build/ rapid prototyping methodology similar to Boehm's Spiral Model (Boehm, 1988). The initial version was delivered to RL in August 1989.

The SLCSE (command executive and tools) is approximately 120K source lines of code. SLCSE consists of approximately 80 percent Ada and 20 percent existing code written in other languages (FORTRAN, MACRO). It is VAX/VMS-based and uses DEC VT100-type terminals. The SLCSE framework was designed so it could be tailored to specific software development projects. SLCSE possesses a generic user interface subsystem and a generic database subsystem that are used to create project-specific environments. The project-specific environments may be thought of as instances of the generic environment framework.

RL initiated the SLCSE Beta testing at various Air Force Logistics Command (AFLC) sites.

The SLCSE test sites included

1. Warner-Robins Air Logistics Center (WR-ALC), Robins AFB, Georgia

   Warner-Robins conducted an evaluation of the SLCSE and its tools. GRC integrated a test toolset into the SLCSE.

2. Ogden Air Logistics Center (OO-ALC), Hill AFB, Utah

   GRC added an interface to a C compiler. The database was populated with information for the Navigational Weapons Delivery System (NWDS) Operations Flight Program for F4 aircraft. A Software Requirements Specification was generated using the SLCSE documentation generation capability.

3. Electronic Systems Division (ESD)/MITRE Corporation, Bedford, Massachusetts

   MITRE performed an assessment of the SLCSE to determine its suitability for Air Force acquisition and software development support. Existing software-requirements specifications were mapped into the SLCSE database.

4. C.S. Draper Laboratory, Cambridge, Massachusetts

   The SLCSE database was populated with requirements, test cases and other information for a segment of the Operation Flight Program for an F-111A aircraft. A Software Design Document was generated. Draper personnel used the Automated Life-Cycle Change Impact Analysis (ALICIA) tool to trace the data model.

5. NASA Houston/MITRE Corporation, Houston, Texas

   Existing Ada simulation software was used as a test case. The SLCSE database was populated with information from this project. A Software Requirements Specification and Software Design Document were generated. This effort was conducted for the Software Technology Branch at NASA.

6. Sacramento Air Logistics Center (SM-ALC), McClellan AFB, California

   The C.S. Draper Laboratory effort described above was done jointly with the Sacramento Air Logistics Center.

2.2 MAJOR CHARACTERISTICS OF EXISTING SLCSE

2.2.1 Multiple User Roles

The SLCSE employs the concept of user roles. Examples of roles include acquisition management, programming, and secretarial. The software tools a user may access depends on his or her role. For example, a programmer would have access to general-purpose tools, like editors, along with compilers, and linkers. A programmer typically would not have access to project management and quality assurance tools. While a secretary would probably have access to only a few tools, such as the mailer, bulletin board, and perhaps the documentation generation tool. This concept of user roles is based on the STARS OCD. The user accesses tools for his or her role through the common user interface. Users may have multiple roles. All roles supported by the SLCSE are listed below.

1. Acquisition Management
2. Project Management
3. Project Administration
4. System Analysis

5. System Integration
6. Software Analysis
7. Programming
8. Software Testing
9. Software Integration
10. Verification and Validation
11. Quality Assurance
12. Configuration Management
13. Software Performance evaluation
14. Post Deployment Software Support
15. Training
16. Mission-Critical Software Support Engineering
17. SLCSE Installation
18. Secretarial

2.2.2 Common Database

The SLCSE contains an Entity-Relationship (ER) database that serves as a repository for system software and project information as well as a medium for intertool information exchange. The SLCSE ER database is constructed on top of the SMARTSTAR commercial, relational database.

2.2.3 Automated Document Generation

The Documentation Generation (DOCGEN) tool is provided within the SLCSE for automated document creation. This tool is flexible—it allows users to construct documents in the format meeting their needs. Construction of the document is directed by Document Generation Language (DGL), an Ada-like language including database queries and formatting statements. The DOCGEN tool retrieves information that is stored in the database and formats it into a LaTeX document.

2.2.4 Common User Interface

The SLCSE user interface provides windowing capabilities available on DEC VT100-type terminals. The VT100 keypad is used to provide screen navigation and selection. The SLCSE user interface permits both menu and keyword operation. In

keyword mode, qualifiers are specified through a menu interface. Menu mode is the recommended operational mode. The SLCSE provides the user with a mechanism for adding windows and menus. This is done by using two software tools developed by GRC—the WINNIE Windowing Package and the Menu Operations Organizer (MOO). WINNIE supports the definition and manipulation of windows. MOO controls the sequencing of windows.

2.2.5 Extensibility

There are three levels of tool integration to the SLCSE.

a. Integration of simple tools

   Simple tools are ones having no qualifiers or parameters. Examples of simple tools include a pretty printer, body stubber, and line counter. CASE tools having their own menu systems may also be integrated in this manner.

b. Integration of User Interface (UI)-Conformant tools

   These conformant tools are tools that use qualifiers or require customized menus. Examples of tools requiring menus are the ALS/N ADAVAX compiler and the NSWC BBoard. Some CASE tools, such as the Ada Test and Verification System (ATVS), should be integrated with customized menus.

c. Integration of Database-Conformant tools

   The SLCSE database by default supports the DOD-STD-2167A life-cycle model. The database may be extended through the modification of existing subschemas and through the definition of additional database subschemas (i.e., entities and relationships) to support project-specific life-cycle models, methodologies, documentation standards, and tools. The integration of tools to the database requires an in-depth knowledge of the database structure. This level of integration is probably best left to GRC software engineers, although it has been done by others, e.g., Draper Laboratory.

In addition to the levels of tool integration discussed above, the user interface may also be tailored for individual users by resizing windows, changing the menu item order, and redefining the VT100 keypad.

2.2.6 Support for Multiple Languages

The SLCSE supports Ada and other programming languages commonly used by the DoD. The SLCSE Version 3.9.2 contains an interface to the DEC Ada compilation system and associated tools. The SLCSE can support any computer language for which there is a compiler that runs on VAX/VMS. The languages are supported by

integrating the appropriate compiler, linker, language-sensitive editor, and supporting life-cycle tools into the SLCSE.

Additional languages that GRC has integrated into the SLCSE include FORTRAN, COBOL, C, and JOVIAL J73. Other languages can be integrated into the SLCSE as required.

2.2.7 Support for Multiple Projects

The SLCSE supports multiple projects over a network of computing resources. The maximum number of projects supported by the SLCSE depends on the available disk space and memory and how the VAX system parameters are set. Access to specific tools may be limited by specific projects. Tool access is defined through the use of the SLCSE Environment Manager (SEM).

2.2.8 SLCSE Toolset

This section lists and describes the tools included with the SLCSE Version 3.9.2. Much of the information in this section was extracted directly from SLCSE tools information provided by GRC (GRC, 1991).

The SLCSE toolset contains two kinds of tools—those developed specifically for the SLCSE and those developed for other purposes. The tools developed specifically for the SLCSE will be described first.

Tools Specific to the SLCSE

a. BaselinER - Supports interactive definition, modification, and reporting of configurations and baselines (including all database elements used in the generation of a formal document).

b. Design Tool - Supports interactive population of the Design Subschema.

c. DOCGEN_2167A - Generates formal documents from the contents of the SLCSE project database for all DoD-STD-2167A Data Item Descriptions. The documents generated are listed below.

1. Computer Resources Integrated Support Document
2. Computer Software Operator's Manual
3. Firmware Support Manual
4. Interface Design Document
5. Interface Requirements Specification

6. Software Design Document
7. Software Development Plan
8. Software Product Specification
9. Software Programmer's Manual
10. Software Requirements Specification
11. Software Test Description
12. Software Test Plan
13. Software Test Procedures
14. Software Test Report
15. Software User's Manual
16. System/Segment Design Document
17. System/Segment Specification
18. Version Description Document

d. DOCGEN_REPORT - Supports definition of customized reports from information stored in the SLCSE database.

e. Mentor_Import - Supports the import of requirements and design information (created with Mentor Analyst/Realtime and Design/Realtime tools) into the SLCSE database.

f. Micro_Import - Supports the import of project management information (created with Macintosh-based Microplanner and More tools)

g. ModifyER - Supports graphical navigation of the SLCSE ER database and modification of its contents.

h. Problem Change Report Processor - Supports identification and tracking of software problem reports through the use of interactive forms.

i. ReportER - Supports graphical navigation of the SLCSE Project database, selection of entities and relationships, and generation of reports describing the contents of the database relative to the selected entities and relationships.

j. Requirements Tool - Supports interactive population of the System_Requirements and Software_Requirements Subschemas of the SLCSE Project database.

k. SDL_Compiler - Translates Schema Definition Language (SDL) source code (a specialized Ada-like language for specifying subschemas and the entities, relationships, and attributes that comprise them) into Structured Query Language

(SQL) statements that are interpreted by SMARTSTAR to create a low-level relational implementation of the ER SLCSE database.

l. SDL_Convert - Translates SDL into PROLOG for use by AnalyzER that performs consistency analysis.

m. SEM - Supports the definition/modification of a site/company/organization environment, and definition/modification of one or more software development projects within the larger environment.

n. Test Manager - Supports interactive population of the Test Subschema

o. VerifyER - Supports consistency checking on user-selected database entities, and produces reports identifying any inconsistencies in relationships between the selected entities.

The tools listed below are those developed for other purposes, but that are part of the SLCSE toolset.

Government Furnished Software (GFS), Public Domain, and GRC Proprietary

a. ADL - The Ada Design Language tool supports text descriptions of software design using a formal, Ada-like, specification language, and generation of reports based on these specifications. ADL is geared toward the development of Ada software.

b. ALICIA - Supports interactive navigation of the SLCSE database, identification of entities for a proposed change, and review of estimated change impact on database entities and relationships. The use of Alicia requires a workstation supporting the Graphical Kernel System (GKS).

c. AMS - The Automated Measurement System tool supports the definition, collection, and reporting of quality metric information based on the RL Software Quality Framework.

d. AnalyzER - Processes the PROLOG (created by the SDL_Convert companion tool) and checks for consistency within the schema definitions (e.g., relationships have both domains and ranges).

e. Kermit - Supports text and binary file transfer between different computers via phone lines or network connections.

f. MOO - The Menu Operations Organizer defines the sequencing of windows. MOO supports the definition of an interactive application's operational structure and serves as a unifying mechanism connecting the interactive windows (constructed via WINNIE) with the user responses directing operation of the interactive application. MOO was developed by GRC.

g. TeX/LaTeX - Companion products that support general-purpose text formatting and printing (used by DOCGEN_2167A & DOCGEN_REPORT).

h. WINNIE - Supports interactive prototyping of window-oriented user interfaces for VT100-compatible terminals, and provides the runtime window management utilities used by the prototypes. It was developed by GRC.

Commercial software to be purchased for use with the SLCSE is discussed in Section 2.3.

2.3 COMMERCIAL SOFTWARE TO BE PURCHASED

The SMARTSTAR relational database is required by the SLCSE and must be purchased. The other software is optional and should only be purchased if needed for a project.

SMARTSTAR - Interface layer between the SLCSE ER database and its underlying hardware (i.e., Sharebase) or software (i.e., DEC Relational Database) relational implementations. A special SMARTSTAR license, for use only with the SLCSE, was purchased by NOSC from GRC.

DEC Ada compiler - The compiler and associated tools are available from Digital Equipment Corporation.

FORTRAN, COBOL compilers - These compilers are available from Digital Equipment Corporation.

JOVIAL J73 compiler - The compiler is supported by Proprietary Software Systems.

Other VMS utilities and tools - Other tools and utilities bundled with VMS or licensed through DEC or third party vendors (e.g., EDT, EVE, MAIL, LSE-ADA, LSE-FORTRAN, LSE-COBOL, LSE-J73, MACRO).

2.4 HARDWARE REQUIREMENTS

The SLCSE Version 3.9.2, installed on the NOSC Code 411 VAX 3100, included the SLCSE source code and the SLCSE toolset described in Section 2.2.8, SMARTSTAR, and the DEC Ada compiler and associated tools. This installation requires 97,770,520 bytes (190960 blocks) of disk storage space. The NOSC Code 411 VAX 3100 contains 16 megabytes of memory. GRC software engineers recommend five megabytes memory as a base for the SLCSE and 4 additional megabytes for each user.

2.5 CHARACTERISTICS OF THE FUTURE SLCSE

A 5-year contract for major enhancements and the products of the SLCSE was awarded by RL to ISSI in August, 1991. Major enhancements are scheduled to be

completed by the end of August, 1993. The remaining 3 years are to provide user support. ISSI refers to the future SLCSE as the Enhanced SLCSE (E-SLCSE). The improvements to be made to the SLCSE are summarized in this section.

2.5.1 User Interface Enhancements

The user interface will be reimplemented to run on top of X Windows and Motif. The existing user interface is VT100-based. The X Window graphical window system, which was developed by the Massachusetts Institute of Technology during the mid-to-late 1980s, offers many advantages. These advantages include improved interoperability because of the use of a client/server architecture across a network, the versatility and flexibility in constructing menus, the use of X Windows by many tool vendors, the ease in using X Window applications, availability of tools for X Window development, support for graphics, as well as increased portability.

2.5.2 Repository Enhancements

The dependence on the commercial relational database, SMARTSTAR, will be eliminated. The enhanced SLCSE will use the ANSI standard SQL as the RDBMS interface layer to COTS RDBMS products. RDBMS products that currently support SQL include Interbase, SQL server, Oracle, Ingres, Informix, Progress, RDB, and Sybase. Other repository enhancements include the development of a graphical user interface for the Schema Design Language (SDL) tool used with the database, the addition of advanced object oriented features, the redesign of ALICIA to include a repository browser, and others.

2.5.3 Tool Enhancements

Each existing SLCSE tool will be analyzed. Some will be reengineered. Some will be ported to POSIX, and others will be replaced with an equivalent POSIX tool.

A full functionality desktop publishing package (e.g., Framemaker, Interleaf) will be integrated into the SLCSE to allow users to create and update documents in a more natural way than is provided by the existing 2167A documentation capability. Other tools will be developed (e.g., Ada-based user interface builder, E-R editor/browser)

2.5.4 Tool Integration Enhancements

The SLCSE will be delivered with an Ada-based user interface builder for easily constructing or modifying X Window and Motif-based user interfaces using a graphical specification (WYSIWYG) paradigm with automatic Ada code generation of the final interface.

2.5.5 Hosting to Additional Platforms

The enhanced SLCSE will be designed to execute on POSIX/X Windows/Ada workstations (e.g., Sun, Apollo, Hewlett-Packard) or a combination of POSIX/X Windows/Ada workstations with a Digital Equipment Corporation (DEC) VAX/VMS back-end system.

2.5.6 Product Plans

The products of the SLCSE include the development of a marketing strategy, establishment of SLCSE user group, preparation of a newsletter, initiation of a SLCSE workshop, production of a commercial quality video describing the program, and other plans.

Specific services provided to customers will include

- 1-800 telephone line technical service
- Professional product training
- On-site installation and support
- Consulting
- On-site demonstrations
- Regular updates and releases
- Others

The enhanced SLCSE will be provided to Government and DoD contractors free of charge, as long as the user buys software support. Sites will be charged for training, custom products, and special services according to commercial-practice fees.

3.0 SLCSE AS A PROJECT TOOL

3.1 SLCSE STRENGTHS

3.1.1 Tailoring of SLCSE Toolset

Probably the greatest strength of the SLCSE is the capability to tailor the SLCSE toolset so it provides the tools required for a specific project. Those tools in the SLCSE toolset that are not needed can be made invisible and inaccessible to a project. More importantly, additional tools needed for a project may be integrated into the SLCSE by users.

Tools a project manager might consider for integration are project-specific tools and others needed but that are not in the SLCSE toolset, such as Ada reverse engineering tools. Sometimes it may be desirable to integrate a tool that has the same or similar functionality as an existing SLCSE tool. Users should be able to use a tool, from within the SLCSE, that they have used extensively and like rather than be forced to use the comparable one from the SLCSE toolset.

The capability to develop custom menus as part of the tool integration process is also a valuable feature. It is easier for a user to execute a tool by filling in a menu than by continually referring to a tool's user's guide.

NOSC has successfully tailored the SLCSE toolset. Tool integration is discussed in more depth in section 4.0 and appendix B.

3.1.2 User Roles

Each person on a project has specific responsibilities. The responsibilities define a person's role on the project. Usually the person's role and responsibilities are defined implicitly, which allows the lines between roles to be blurred. The concept of "user roles" from the STARS OCD explicitly defines the roles and their responsibilities.

The design decision by GRC and the Air Force to employ the concept of user roles has increased the utility of the SLCSE. This is an attribute that production quality SEEs should have. When employing the user-role concept each person on a project is assigned one or more roles. Members of each role have access to a specific set of tools. Limiting tool access helps reduce the occurrence of software disasters.

3.1.3 Ease of Learning

The SLCSE was easy to learn. The people who attended the SLCSE training at NOSC had no difficulty learning the system or its user interface. Learning the intricacies of some tools in the SLCSE toolset was more difficult.

3.1.4 Document Generation Capability

The document capability of the SLCSE allows the user to develop all documents required by 2167A from within the environment. All of the information needed to automatically generate the required document is contained within the SLCSE database, making it much easier to generate these documents. This is an extremely powerful feature of the SLCSE.

3.1.5 Stability

The SLCSE is a stable system. During the 4 months of NOSC's investigation of the SLCSE, no user caused the system to crash or have other downtime. As many as five users used it simultaneously. All users felt the response time was adequate.

3.1.6 Tailoring of SLCSE Database

The SLCSE database can be tailored to support different entity relationships depending on the needs of the project. Database tailoring is done outside of the SLCSE by the database administrator. Tailoring is done by editing the schema definition files.

3.2 SOME SLCSE WEAKNESSES

3.2.1 User Interface

The main weakness of the SLCSE is its user interface. The user interface has a number of features that should be fixed in the next version. The SLCSE provides a number of excellent capabilities, but the user has to interact with the interface to reach them. Most users will not use an environment with a difficult interface even though it provides outstanding capabilities.

Among the user interface problems is no fast way to return to the top-level menu. For example, when a user works down through four levels of menus and diagrams using a document generation tool, each menu and diagram must be exited separately before a new activity can be started.

Returning to the top level is even worse because the method of exiting each menu varies. The top-level menu is exited by moving the cursor to the menu option EXIT and then pressing the return key. Other menus are exited in a similar fashion only DONE is the menu option. Still other menus are exited by typing a keypad 0. During NOSC's use of the SLCSE, the wrong method was almost invariably the first method tried.

In the tools menu, the user can move to a particular tool either by moving the cursor up or down the list with the arrow keys or by typing the tool's number from the

list which moves the cursor directly to the tool. The files in the object menu are also numbered. The numbers cannot be used to move quickly to the desired file, however. Since the object menu can become lengthy, this would be a helpful capability.

Information messages from the tools are transmitted to the user by writing them in a window of the SLCSE screen. Frequently, the messages are displayed for such a short time the user is not sure whether they are error messages or normal completion messages. For example, when using the ALS/N ADAVAX compiler the message flashed by so quickly that a user could not tell whether the message was "Fatal Error detected - compilation aborted" or "ADAVAX: Normal successful completion."

3.2.2 User's Manual

The SLCSE interface problems are exacerbated by not having a user's manual. Usually, the only way to get help with problems is to call GRC. While GRC representatives have always been helpful, calling them requires a maintenance agreement for day-to-day use of the SLCSE. Most of the questions NOSC asked could have been answered by a comprehensive user's manual.

The tool integration user's manual (GRC, 1989) is neither complete nor accurate. In many ways this is more frustrating than having no manual. Both WINNIE and MOO user's manuals (Cooper, 1986) and (Lamb, 1989) are also required to complete the tool integration process. Even with all three manuals, help from experienced GRC personnel was required to complete integration of the BBoard tool into the SLCSE. GRC provided the report, "SLCSE Site Specific Tool Integration" (GRC, 1990), that was far more helpful in this process than all three of the user's manuals.

3.2.3 Error Messages

The SLCSE notifies the user when internal error conditions occur. These messages frequently are not clear, which makes it difficult for the user to understand what is wrong and decide how to correct them. The SLCSE On-The-Job-Training Manual (GRC, 1991) provides an incomplete list of the error messages. However, a complete list is not provided.

3.3 SUGGESTED ENHANCEMENTS TO SLCSE

Here are suggested enhancements to the SLCSE:

1. The ability to execute VAX/VMS commands from within the SLCSE. There are some VMS commands that can only be executed from outside of the SLCSE. Executing them requires exiting the SLCSE, executing the command, and then re-entering the SLCSE.

2. A common method of exiting menus. Section 3.2.1 discusses why this is a desirable feature.

3. A simple way to jump back to the top level menu. Section 3.2.1 discusses the drawbacks of the current method.

4. The ability to display categories of tools. When a user has access to a number of tools the user must traverse the entire list to find a particular tool, or memorize the tool numbers used most frequently. The user should be given the ability to display tools by categories, for example, VMS tools, Ada development tools, etc.

5. Users need more help when they get error messages. The SLCSE error messages are not usually self-explanatory nor is there a manual containing an explanation of the messages. More meaningful error messages and a manual containing all error messages with suggested actions would be very helpful.

6. The SLCSE installation process needs to be simplified. The current installation process is so complex that only GRC can do it. A production quality SEE must be installable by users or the on-site system administrator.

7. The ability to print the SLCSE screens is needed. Currently there is no way for users of the SLCSE to capture screens for inclusion in their user documentation and presentation materials. This feature is needed, for example, when preparing documentation on how to use the customized menus for ALS/N tools.

8. The instructions for tool integration need to be simple, precise, and accurate What is available is helpful, but it needs to be expanded and corrected. Currently the instructions are spread over a number of documents.

4.0 SLCSE TOOL INTEGRATION

Tool integration is the process of inserting tools into the SLCSE so they appear in and may be accessed from the SLCSE tools menu. Tool integration is done under configuration control. Only the SLCSE system manager or someone with system privileges may add tools. The tool selection to be integrated is a management decision. Project-specific and any other appropriate tools should be integrated.

The tool integration capability encourages use of appropriate tools by project personnel. This capability also allows users to build customized menus that simplify tool use.

4.1 LEVELS OF TOOL INTEGRATION

The tool integration process varies depending on the level of tool integration. Three levels based on how deeply the tool integrates into the SLCSE are simple tool integration, UI-conformant tools, and database conformant tools.

4.1.1 Simple Tools

Simple tools do not contain qualifiers and parameters and do not require a setup window. A setup window is a menu in which the user provides information needed to run the tool. When tool integration is completed the tool name appears in the SLCSE tools menu along with the other SLCSE tools. An example is Pretty Printer, a simple tool, which has no qualifiers and requires no input prior to commencing operation.

In addition to ALS/N tools, NOSC integrated the following simple tools into the SLCSE during the investigation:

1. Ada Primitive Order Compilation Order Tool (APRICOT)
2. Bit-Oriented Message Definer (BMD)
3. LGEN: A Language Generator Tool
4. File_Checker
5. Pretty Printer
6. Body Stubber

Tool descriptions are found in appendix D.

4.1.2 UI-Conformant Tools

UI-conformant tools contain qualifiers or parameters and require a setup window (customized menu) but do not access information from or provide information to the

SLCSE database. The SLCSE supports the development of these setup windows by providing two table-driven menu or window development tools, WINNIE and MOO. An example is the AD4VAX compiler. At a minimum a UI-conformant tool requires a file name be provided before execution commences. It usually has a number of qualifiers set to provide optional results.

Customized menus help by allowing users to choose and invoke options that are displayed within the menu. This capability means users will spend less time referring to tool-user guides. The integration of tools with qualifiers and parameters is considerably more work than integrating simple tools. In addition to using WINNIE and MOO the user must write an Ada procedure that reads in the user specified data from the customized menu and builds the appropriate DCL command. When tool integration is completed the tool name appears in the SLCSE tools menu.

In addition to ALS/N tools, NOSC integrated the following UI-Conformant tools into the SLCSE:

1. Electronic Bulletin Board (BBoard) BBoard
2. LEXGEN: A Lexical Analyzer Generator Tool

Tool descriptions are found in appendix D.

4.1.3 Database Conformant Tools

Database conformant tools interface directly with the SLCSE project databases. Database conformant tools are usually built specifically for inclusion into the SLCSE and were not considered during this study. An example is the ReportER, that is a default tool of the SLCSE.

For the SEE prototypes task simple tools and those with qualifiers and parameters were integrated. No tools were integrated to the database.

4.2 HIGH-LEVEL STEPS FOR TOOL INTEGRATION

Figure 4-1 illustrates the high-level steps followed to integrate a simple tool and a UI-conformant tool. The figure is taken from the GRC report (1989). Detailed explanations of the required steps are contained in appendix B. The steps for database conformant tool integration will not be discussed.

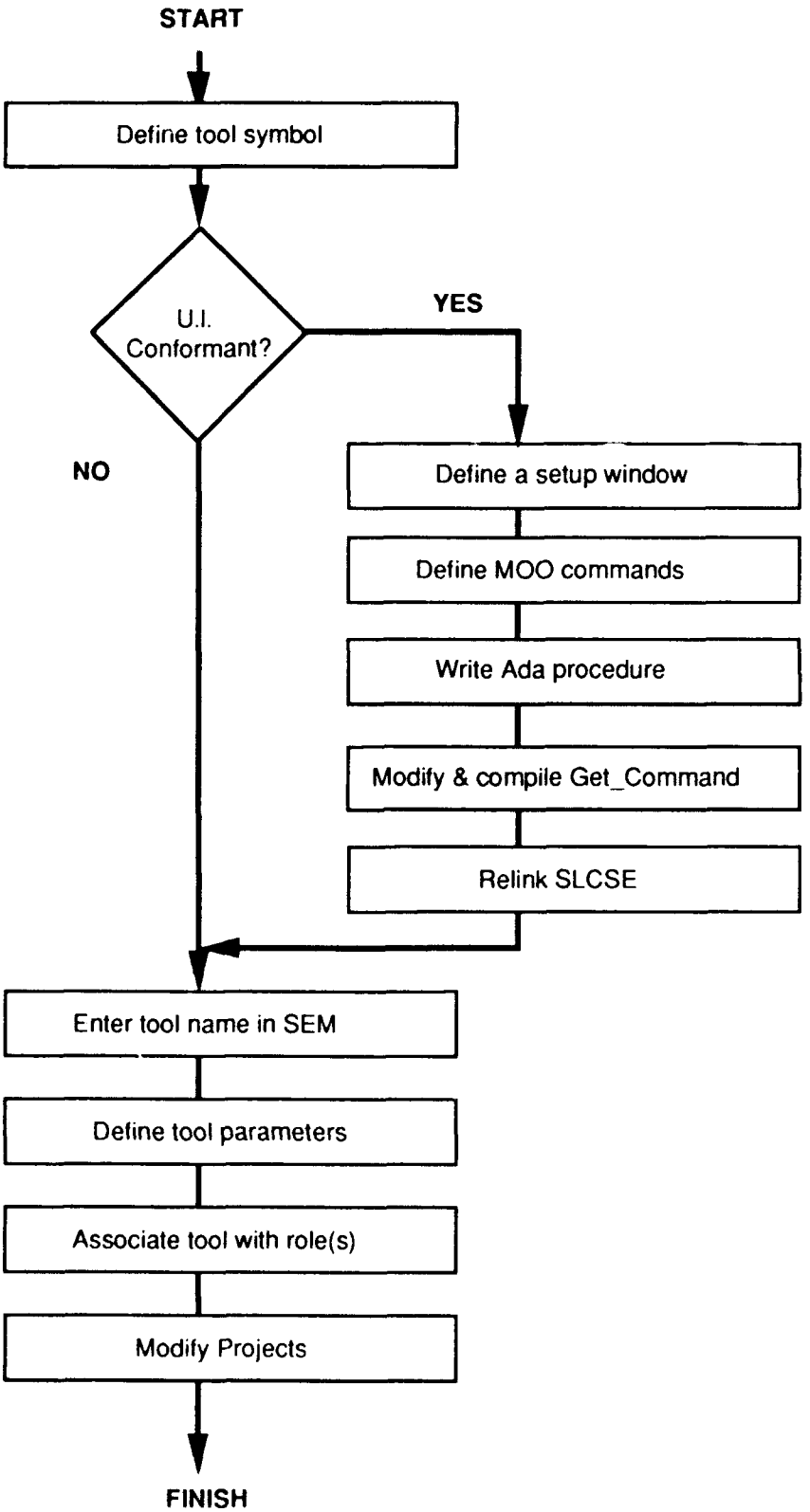


Figure 4-1. Tool integration process.

4.2.1 Integrating Simple Tools

The process of integrating a simple tool is a subset of the process for integrating a UI-conformant tool (figure 4-1). The integrator needs to define a command alias to be used as the tool symbol, for example, defining the alias BMD to be equivalent to the command "run bmd." The tool symbol is defined in one of the SLCSE system startup command files. Then the tool must be added to the SLCSE using the SLCSE Environment Manager (SEM). This requires entering the tool symbol into the tools list in SEM and answering questions about the way the tool runs. For example, does the tool output to the screen or to a file, will it run in batch or interactive mode, etc. Then the tool must be associated with the role or roles that will require access to tool. Finally, the project(s) currently running under the SLCSE must be modified to recognize the tool's existence.

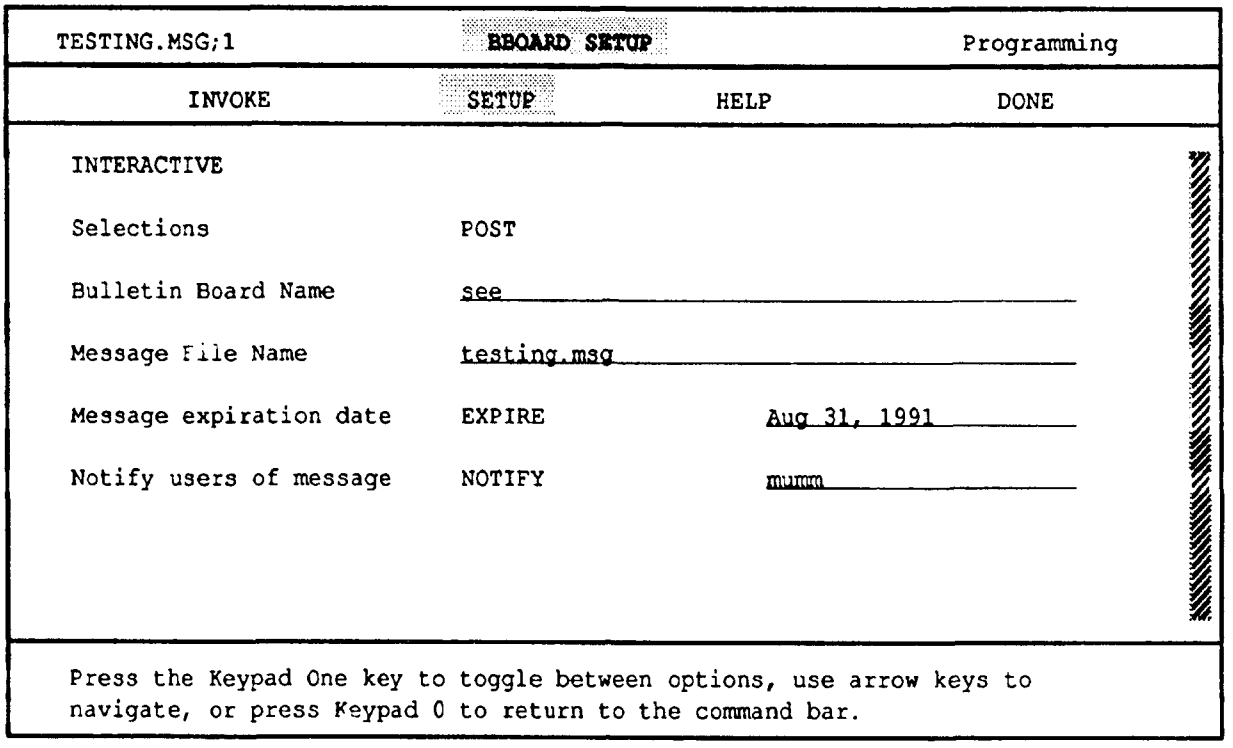
4.2.2 Integrating UI-Conformant Tools

Integrating UI-conformant tools is a more difficult and time-consuming task than integrating simple tools. This difficulty is because of the necessity to construct a setup window. The setup window defines the required qualifiers and allows entry of any required parameters. NOSC personnel defined the setup window for the BBoard tool. The window selections field can be toggled to each possible BBoard action, i.e., it can take the values post, create, help, read, status, or garbage (collection). Figure 4-2 shows the setup windows for each. The contents of the setup window vary with the value in the selection field. A brief description of the BBoard is provided in appendix D. Constructing and interpreting the setup window requires the use of an Ada compiler and three tools provided with the SLCSE.

First, the tool integrator must define either a global symbol or DCL command for the tool. After the tool symbol is defined, the setup window must be constructed using the WINNIE windowing tool written by GRC. Then MOO, also provided by GRC, must be used to define the sequencing of responses within the setup window. Neither WINNIE nor MOO commands are easy for the first-time user even with the users manuals (Cooper, 1986 and Lamb, 1989). (Excerpts of the WINNIE and MOO commands for the BBoard setup window are given in appendix B in figures B-4 and B-5.) Next, an Ada procedure must be developed and compiled. This procedure takes the information the user provides through the setup window and creates the required command to be passed to the VMS operating system. The procedure which drives the execution of site-specific tools must also be modified and recompiled. Then, the integrator must relink the SLCSE. Finally, the tool must be added to the SLCSE using the SEM. This process is the same one followed when integrating a simple tool.

4.3 LESSONS LEARNED

The first time integrating either type of tool is difficult and nonintuitive. After the first tool of a specific type (simple or UI-conformant) has been integrated, repeating the process is not particularly difficult. Integrating UI-conformant tools remains time-consuming. For example, the first UI-conformant tool integrated by NOSC personnel (BBoard) took approximately 34 hours of work. The second UI-conformant tool required approximately 5 hours, while integrating simple tools takes less than an hour.



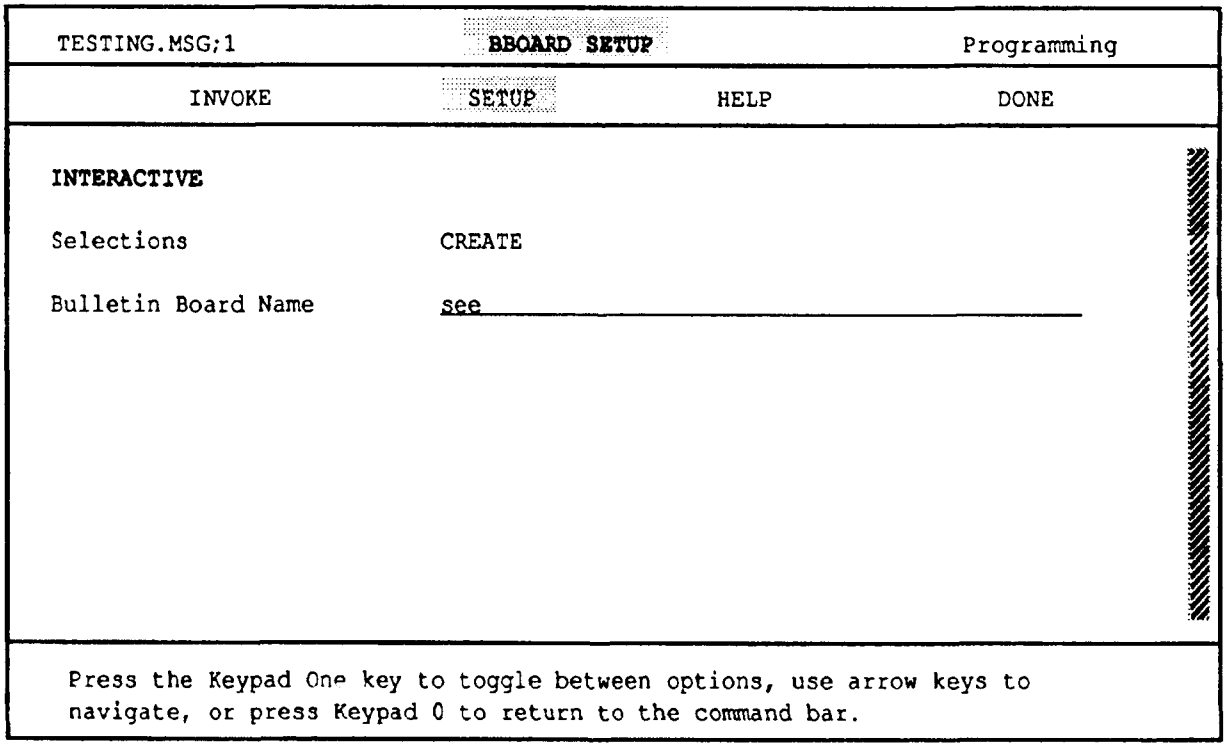
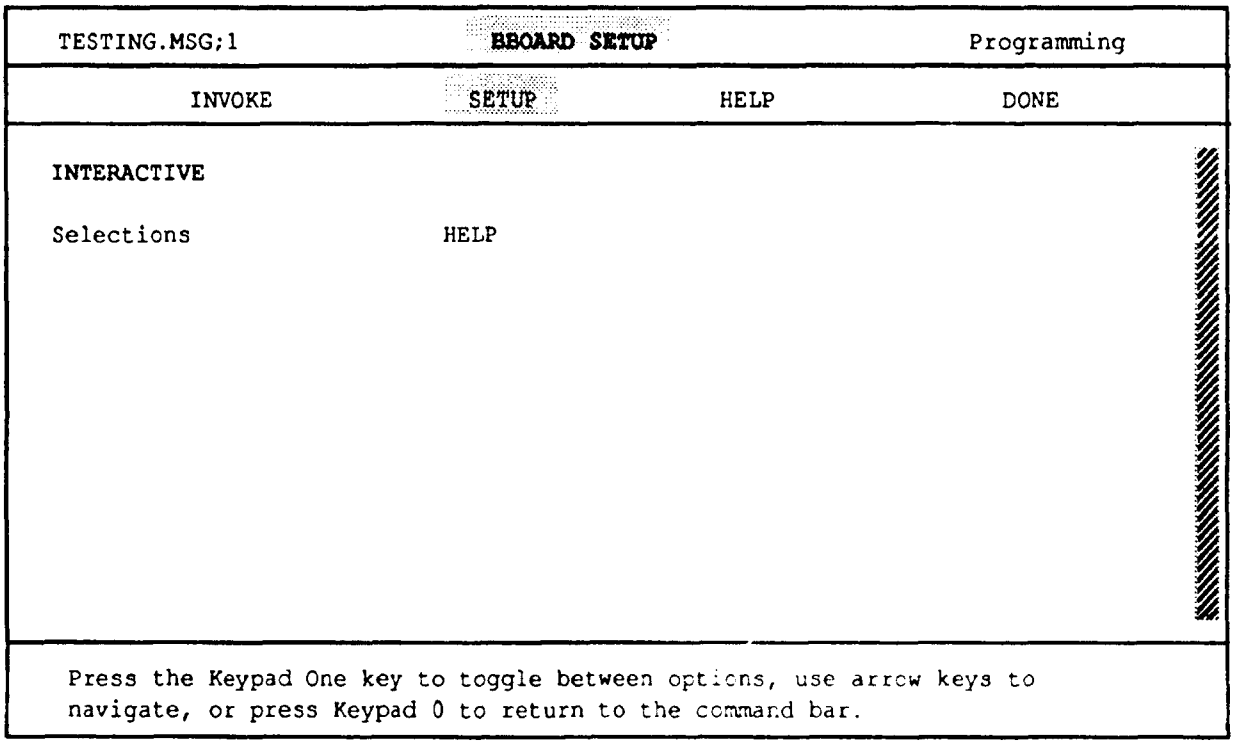


Figure 4-2. BBoard setup window.



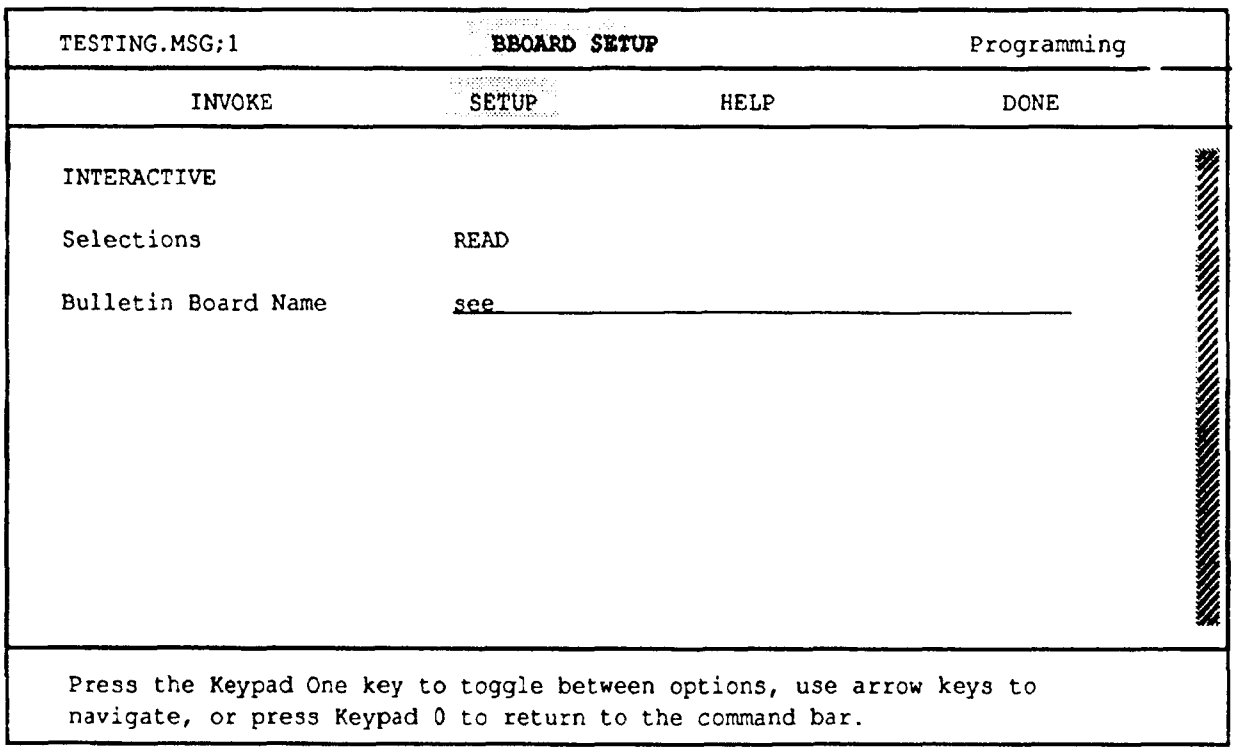
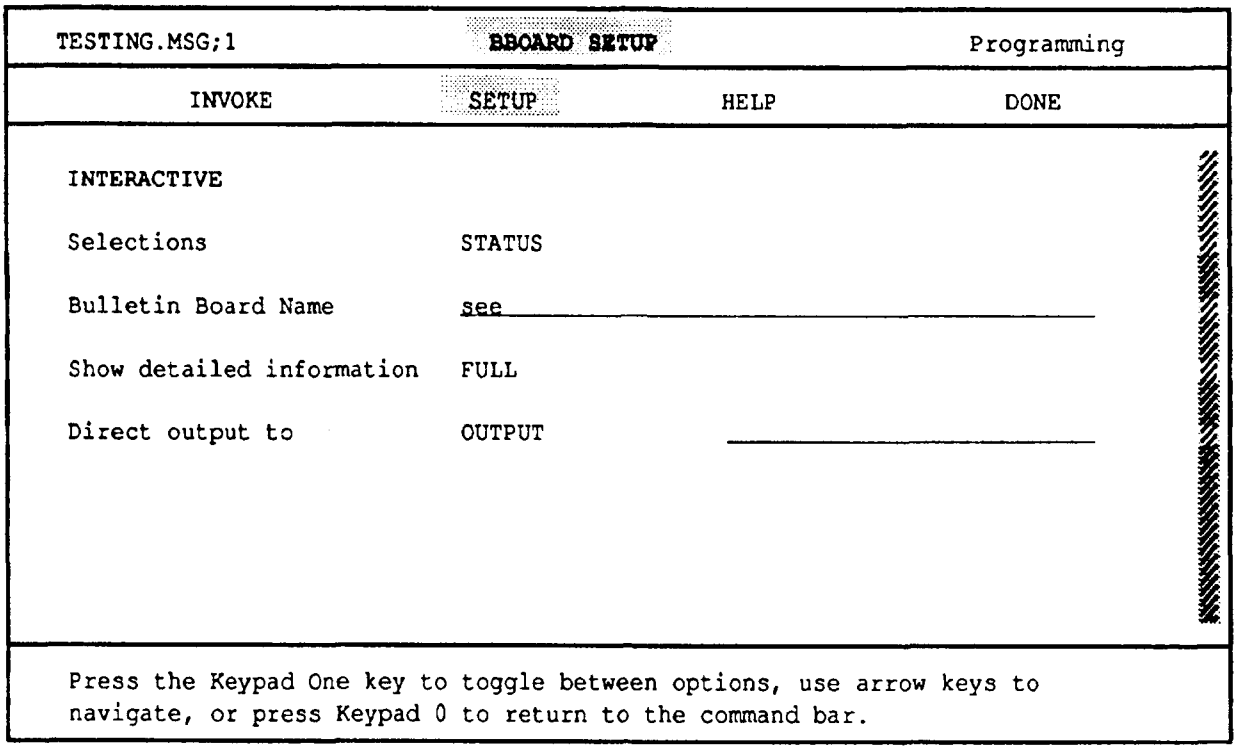


Figure 4-2. BBoard setup window (continued).



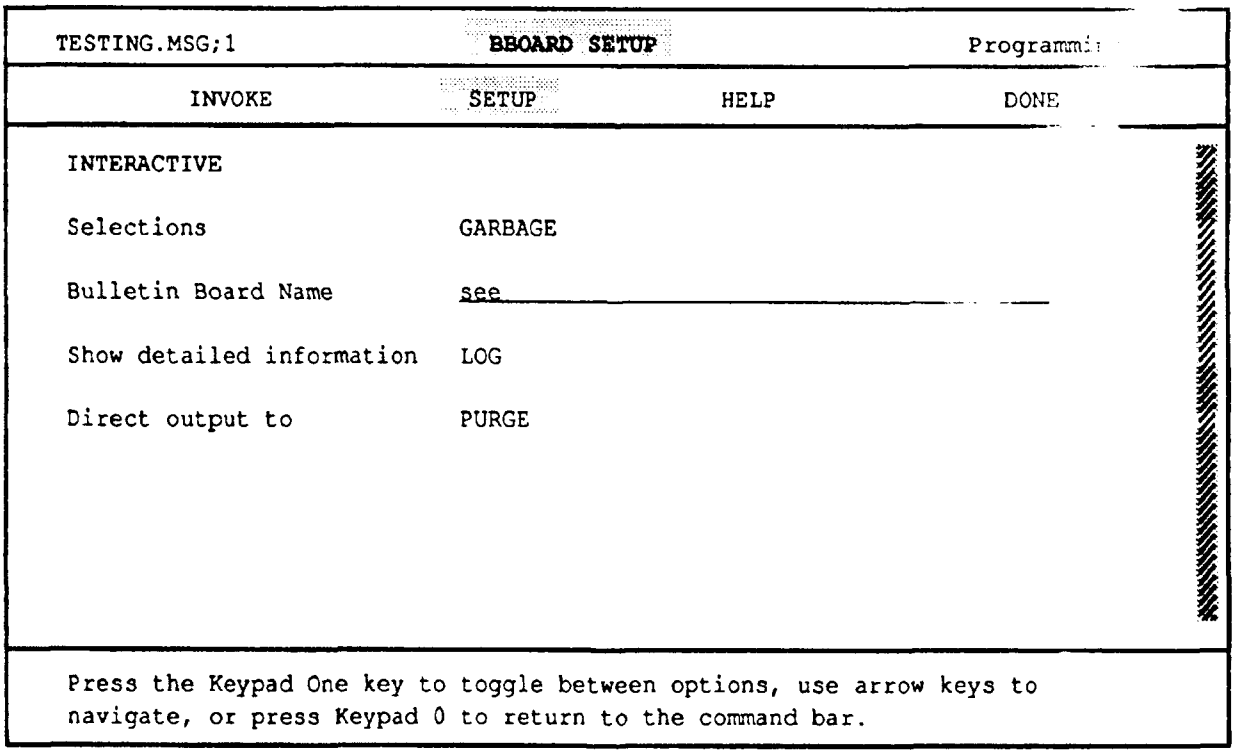


Figure 4-2. BBoard setup window (continued).

5.0 INTEGRATING ALS/N TOOLS

5.1 INTEGRATION

The integration of the ADAVAX compiler Version 4.3 and associated tools was accomplished by following steps discussed in Section 4.0. The two ALS/N library management tools listed below were integrated into the SLCSE by following the steps for simple tool integration described in Section 4.2.1. No setup windows were required.

a. CLIB - Command Interface

b. SLIB - Menu Screen Interface

The four tools shown below were integrated by following the steps for UI-conformant tool integration described in Section 4.2.2.

a. ADAVAX - ADAVAX Compiler for the VAX/VMS Target

b. LNKVAX - Linker for the VAX Target

c. EXPVMS - Exporter to VAX/VMS Target

d. IMPVAX - Importer to VAX/VMS Target

5.2 ALS/N SETUP WINDOW DESCRIPTIONS AND USE

Figures 5-1 through 5-4 show the setup windows for ADAVAX, LNKVAX, EXPVMS, and IMPVAX. These windows were created to provide all the options described in the ALS/N reference handbook (NAVSEA, 1989). Each qualifier for these four tools may be set on or off. Toggling between on and off is done by pressing keypad 1. After a tool is invoked and it completes, the user may examine error messages and information pertaining to the options selected by pressing keypad 3.

ADAVAX Setup Window

Figure 5-1 shows the setup window for the ADAVAX compiler. The top screen of the figure, third line from the top, shows the "INTERACTIVE" and "BATCH" compilation capability. In the figure this option is set to "INTERACTIVE." When an interactive or batch compilation completes successfully the following message is displayed on the user's monitor:

ADAVAX : normal successful completion

The user must give the file name to compile. The remainder of the top screen shows the listing control qualifiers that may be selected.

The bottom screen of Figure 5-1 shows the special processing qualifiers and the first three of the special compilation unit qualifiers.

```
                              ADAVAX SETUP                          Programming

      INVOKE                  SETUP                 HELP                 DONE

  INTERACTIVE                                       UPDATE WITH SELECTED OBJECT

  Filename to Compile  _____________________________________________

  Listing Control Options

  Produce Symbol Attribute Listing              NO_ATTRIBUTE
  Produce Diagnostic Summary Listing            NO_DIAGNOSTICS
  Produce Machine Code Listing                  NO_MACHINE_CODE
  Include Diagnostics of Note Severity          NO_NOTES
  Produce Ada Source Listing                    SOURCE
  Produce Summary Diagnostics Listing           NO_SUMMARY
  Produce Cross-Reference Listing               NO_CROSS_REFERENCE
  Include Private Specs in Listing              PRIVATE

  Press <Return> to advance to the Setup window, use arrow keys to navigate.
```

```
                              ADAVAX SETUP                          Programming

      INVOKE                  SETUP                 HELP                 DONE

  Special Processing Options

  Provide Run-time Error Checking               CHECKS
  Generate Code if Warning Diagnostics          CODE_ON_WARNING
  Produce Container if Severity Permits         CONTAINER_GENERATION
  Generate Debugger Symbols & Code              DEBUG
  Monitor Subprogram Execution Frequency        NO_MEASURE
  Enable Global Optimization                    NO_OPTIMIZE
  Provide Calling Sequence Traceback            TRACE_BACK

  Special Compilation Unit Options

  Activate All Compiler Options Below           NO_COMPILER_MAINT
  Compile Generic Built-in Subprograms          NO_BIS_COMPILE
  Compile New ADA_RSL Package Spec              NO_RSL_COMPILE

  Press the Keypad One key to toggle between options, use arrow keys to
  navigate, or press Keypad 0 to return to the command bar.
```

Figure 5-1. ADAVAX setup window.

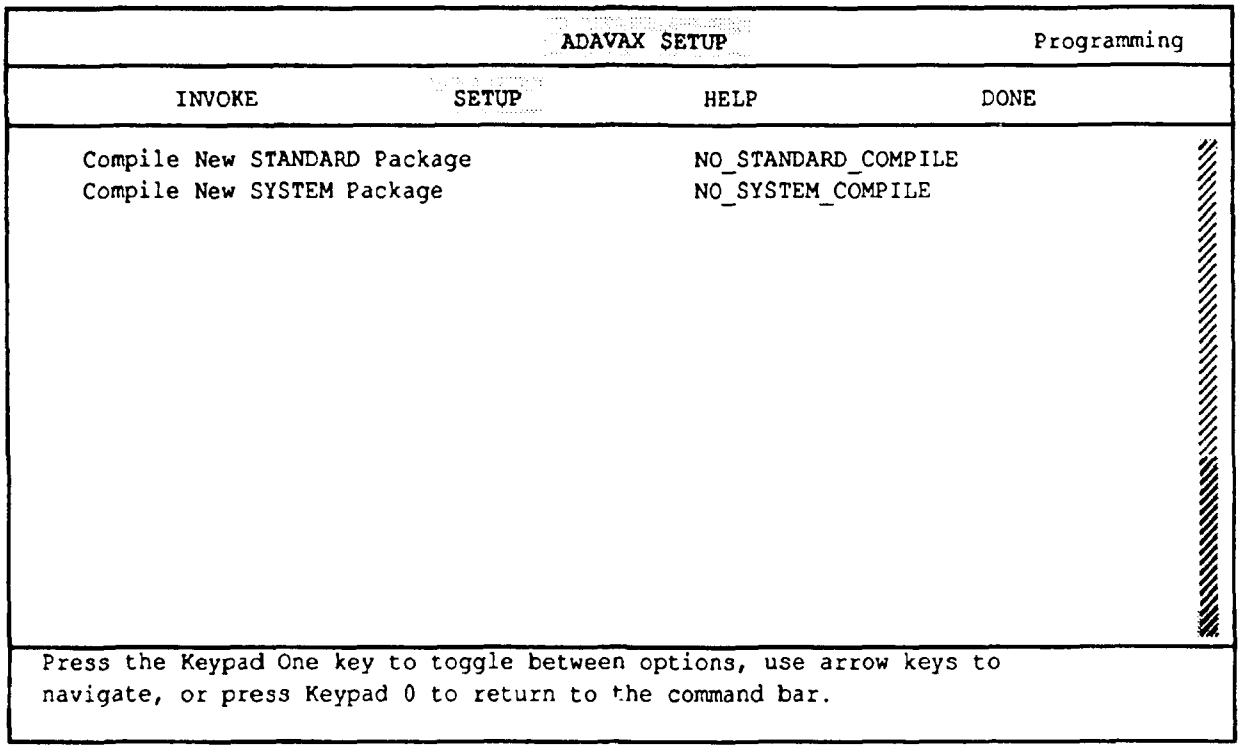


Figure 5-1. ADAVAX setup window (continued).

```
                              LNKVAX SETUP                  Programming
              INVOKE           SETUP           HELP           DONE

  INTERACTIVE

  Main Subprogram                    ______________________________
  Output Container                   ______________________________
  Unit List Filename                 ______________________________
  Produce Unit Listing                        UNITS
  Produce Symbol Listing                      SYMBOLS
  Produce Elaboration Order Listing           ELAB_LIST

  Special Processing Options

  Produce Container for Debugging             NO_DEBUG
  Produce Container for Performance Measure   NO_MEASURE
  Permit Partial Container Creation           NO_PARTIAL
  Link All Referenced Units                   SEARCH

 Press <Return> to advance to the Setup window, use arrow keys to navigate.
```

```
                              LNKVAX SETUP                  Programming
         INVOKE           SETUP           HELP           DONE

  Maintenance Options

  Propagate Linker Stack Dumps                NO
  Produce Functional Trace of Execution       NO
  Produce Trace of Data Transactions          NO

 Press the Keypad One key to toggle between options, use arrow keys to
 navigate, or press Keypad 0 to return to the command bar.
```

Figure 5-2. LNKVAX setup window.

```
                              EXPVMS SETUP                          Programming

      INVOKE                  SETUP                 HELP                 DONE

INTERACTIVE

Linked Container      _____________________________________________________
Export Module         _____________________________________________________
Directive File        _____________________________________________________

Produce Program Sections Map Listing          NO_MAP
Produce Symbol Listing                        NO_SYMBOLS

Special Processing Options

Report Elapsed CPU and Wall Clock Time        NO_ACCOUNTING
Allow Use of Symbolic Debugger                NO_DEBUG
Perform Frequency Analysis                    NO_MEASURE
Produce Symbols List for Debugger             NO_DEBUG_SYMBOLS

Press <Return> to advance to the Setup window, use arrow keys to navigate.
```

```
                              EXPVMS SETUP                          Programming

      INVOKE                  SETUP                 HELP                 DONE

Maintenance Options

Propagate Exporter Stack Dumps                NO
Produce Functional Trace of Execution         NO
Produce Trace of Data Transactions            NO

Press the Keypad One key to toggle between options, use arrow keys to
navigate, or press Keypad 0 to return to the command bar.
```

Figure 5-3. EXPVMS setup window.

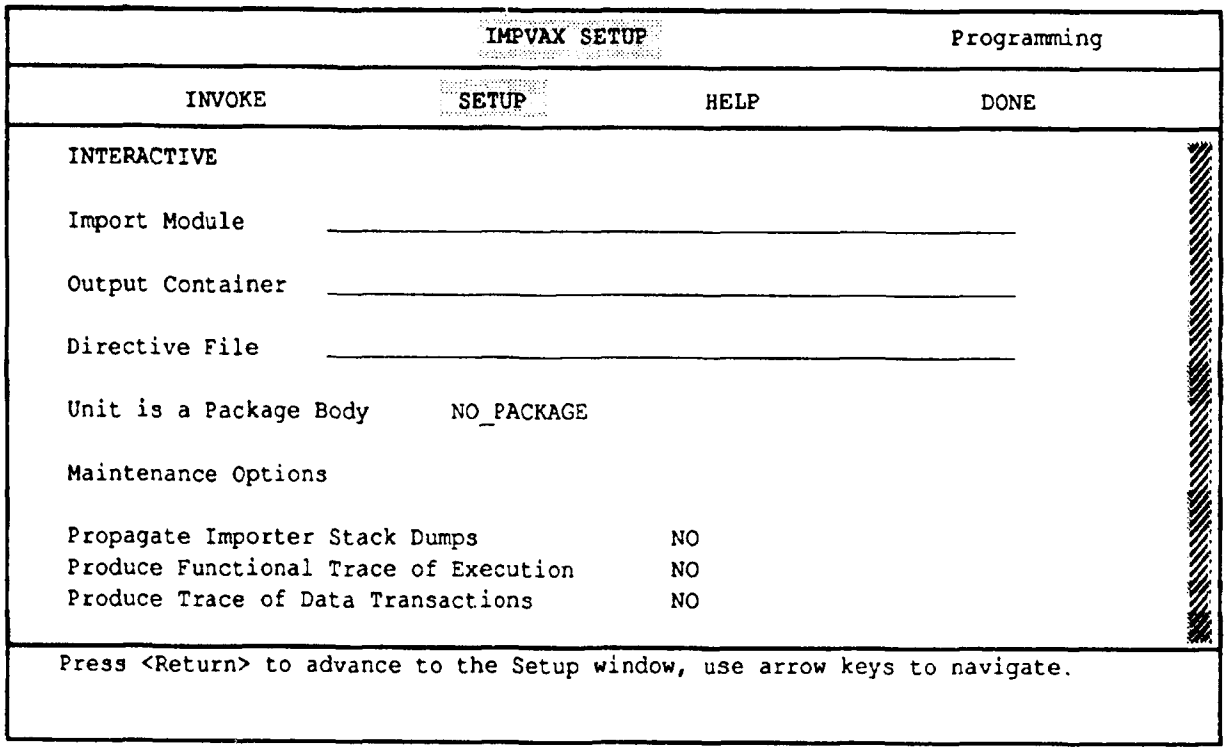


Figure 5-4. IMPVAX setup window.

Figure 5-1 (Continued) shows the remaining two special compilation unit qualifiers.

The three screens shown in Figure 5-1 are the ADAVAX setup window. In this document three pictures are required to show all the qualifiers which are made visible by scrolling.

LNKVAX Setup Window

Figure 5-2 shows the setup window for LNKVAX. In the top window, the first user qualifier is the "INTERACTIVE" and "BATCH" capability. When an interactive or batch link job completes successfully the following command is displayed on the user's monitor:

LNKVAX : normal successful completion

When running LNKVAX the user provides the name of the main subprogram and the output container. The unit list filename is only required when the main subprogram is null. The user sets any of the LNKVAX qualifiers to on or off. These qualifiers include three listings qualifiers, four special processing qualifiers, and three maintenance qualifiers.

EXPVMS Setup Window

Figure 5-3 shows the setup window for EXPVMS. The exporter may be executed either in "INTERACTIVE" or "BATCH" mode. Upon successful completion an interactive or batch execution the following message is displayed:

EXPVMS : normal successful completion

The user provides the name of the linked container and the export module (executable). The directive file is optional. Each EXPVMS qualifier may be set to on or off. These qualifiers include two listings qualifiers, four special processing qualifiers, and three maintenance qualifiers.

IMPVAX Setup Window

Figure 5-4 shows the setup window for IMPVAX. The importer may be executed in either "INTERACTIVE" or "BATCH" mode. Upon successful completion of an interactive or batch execution the following message is displayed:

IMPVAX : normal successful completion

In the setup window the user must give the name of the import module (file containing the import module) and output container. The directive file must be provided when the output container contains a package body. The directive file supplies an entry

point and reference information about the file being imported. Qualifiers include specifying whether a unit is or is not a package body and three maintenance qualifiers.

Appendix C shows the four ALS/N tool setup window definitions using WINNIE, the MOO commands for window sequencing, and the Ada procedures for interpreting the setup windows.

6.0 RECOMMENDATIONS

The SLCSE is a proof-of-concept environment for improving the development process for Navy software. The SLCSE must be improved so SLCSE is of the quality required for Navy software development.

6.1 USER INTERFACE

Section 3.0 states most of the problems with the SLCSE are in the user interface. ISSI needs to make the interface as easy to use as possible. Potential users will not use the environment unless the interface is superb. We recommend that once an early UI prototype is operational a human factors psychologist examine it to make recommendations for improvements. We have found this helpful at NOSC.

6.2 EXTENSIBILITY

One future goal of SLCSE should be to make SLCSE extensible. An environment that can be extended easily is preferable to one that includes a multitude of tools but is only extensible with extreme user effort. Future versions should have a strong capability for integrating tools, including the capability to create customized menus. This capability must be provided to users. It is unsatisfactory for only the environment developer to have the capability to integrate tools. The tool integration process needs to be easier in the future SLCSE.

6.3 DOCUMENTATION

Future SLCSE documentation must be improved. We recommend users' guides, one for tool integration, be written by personnel who are not members of the SLCSE development team. People who are too close to the product tend to write instructions that inadvertently assume the reader has the same knowledge. For example, key details may be glossed over and the project jargon may not be explained. Users need an error message manual that lists all error messages, explains the problem, and suggests solutions.

6.4 E-SLCLSE AS NAVY C^3 SOFTWARE DEVELOPMENT ENVIRONMENT

We recommend first, NOSC take an active role in the new Air Force procurement by participating in reviews, attending demonstrations, and reviewing relevant documents. Members of the SEE Prototypes task can provide insight into needed enhancements including the design of the new UI. This will help ensure the new SLCSE is of the quality required by the Navy.

Second, if the enhanced SLCSE is of the production quality needed by the Navy, we recommend that NOSC develop a SLCSE for Navy C^3 software development. Support of this environment can include an interface to the ALS/N.

6.5 ADDITIONAL ON-LINE CAPABILITIES

We recommend the future SLCSE include an on-line user's guide and error message manual accessible from within the SLCSE. When using windows the user can then see the error message in one window and look it up in the manual in the other.

7.0 REFERENCES

Boehm, B. 1988. "A Spiral Model of Software Development and Enhancement," *IEEE Computer*.

Cooper, D. 1986. *WINNIE GRC Windowing Package* RM-2563/2, General Research Corporation, Santa Barbara, CA.

General Research Corporation. 1989. "SLCSE Software User's Manual Vol. II, SLCSE Environment Manager," Santa Barbara, CA.

General Research Corporation. 1990. "SLCSE Site Specific Tool Integration", Santa Barbara, CA.

General Research Corporation. 1991. "Software Life-Cycle Support Environment (SLCSE) On-the-Job Training Course, Volume II: User Orientation," Santa Barbara, CA.

General Research Corporation. 1991. "Software Life-Cycle Support (SLCSE) Tools," list, Santa Barbara, CA.

International Software Systems Incorporated. 1991. "Proposal for the Software Life-Cycle Support Environment (SLCSE) Enhancements and Demonstrations Program," Austin, TX.

Lamb, J. 1989. General Research Corporation, "Menu Operations Organizer (MOO) Overview," Santa Barbara, CA.

Madden, L., K. Schumaker, and B. Meyers. 1989. "An Electronic Bulletin Board (BBoard) Program." Technical Report 89-03, Naval Surface Weapons Center, Dahlgren, VA.

Meyers, B., and A. Smith. 1988. "LGEN: A Language Generator Tool." Technical Report 88-01, Naval Surface Weapons Center, Dahlgren, VA.

Mumm, R., and S. Parker. 1990. "BMD/Ada Bit-Oriented Message Definer." Technical Report 1384, Naval Oceans Systems Center, San Diego, CA.

Naval Sea Systems Command. 1989. "Ada Language System/Navy Reference Handbook," NAVSEA #ALSN-HBK-PSE-REFHB, Version 3.0.

Smith, A., and B. Meyers. 1989. "LEXGEN: A Lexical Analyzer Generator Tool." Technical Report 89-05, Naval Surface Weapons Center, Dahlgren, VA.

Software Technology for Adaptable Reliable (STARS) Joint Program Office. 1985. "STARS-SEE Operational Concept Document (OCD)," Proposed Version 001.0.

8.0 BIBLIOGRAPHY

Cooper, D. 1989. "Addendum to RN-2563/2 WINNIE Windowing Package," General Research Corporation, Santa Barbara, CA.

Baldwin, R., and D. Emery. 1991. "Technology Assessment of the Software Life-Cycle Support Environment," Contract No. F19628-89-C-0001, MITRE Corporation.

C.S. Draper Labs, 555 Technology Square, Cambridge, Massachusetts. "Software Design Document for the Mission Computer Operational Flight Program Computer Software Configuration Item," Contract No. A12345-89-X-0000, Prepared for Sacremento Air Logistics Center, NMETI, MacClellan AFB, CA.

OFP Program Development, Maintenance Support Section (MASHD), AISF Support Bldg 1202 00-ALC, Hill Air Force Base. "Software Requirements Specification for the RF/F-4 NWDS OFP Computer Software Configuration Item of the F-4 Weapon Delivery System," Contract No. 654-26-0890, Prepared for Hill Logistics Center, MAS/OO-ALC, Hill Air Force Base, UT.

Rome Laboratory, Griffiss Air Force Base, New York. Statement of Work for Software Life-Cycle Support Environment (SLCSE) Enhancements and Demonstration Program, PR NOS. B-1-3321/B-1-3368, 21 Dec 90.

APPENDIX A: SLCSE INSTRUCTIONS FOR THE NOVICE

This appendix describes many basic operations needed when using the SLCSE. The appendix is intended for the new and novice SLCSE user. Figure A-1 shows the top-level SLCSE screen. The menu bar is the second window from the top. These instructions will frequently refer to this menu bar. It contains the primary choices the user has the top level of the SLCSE. The choices are OBJECTS, TOOLS, SETTINGS, HELP, and EXIT. While a selection in the menu bar is highlighted, pressing return will either cause a pull down menu to appear for the user's next choice, i.e., OBJECTS or TOOLS; or it will cause the SLCSE to perform that action, i.e., EXIT.

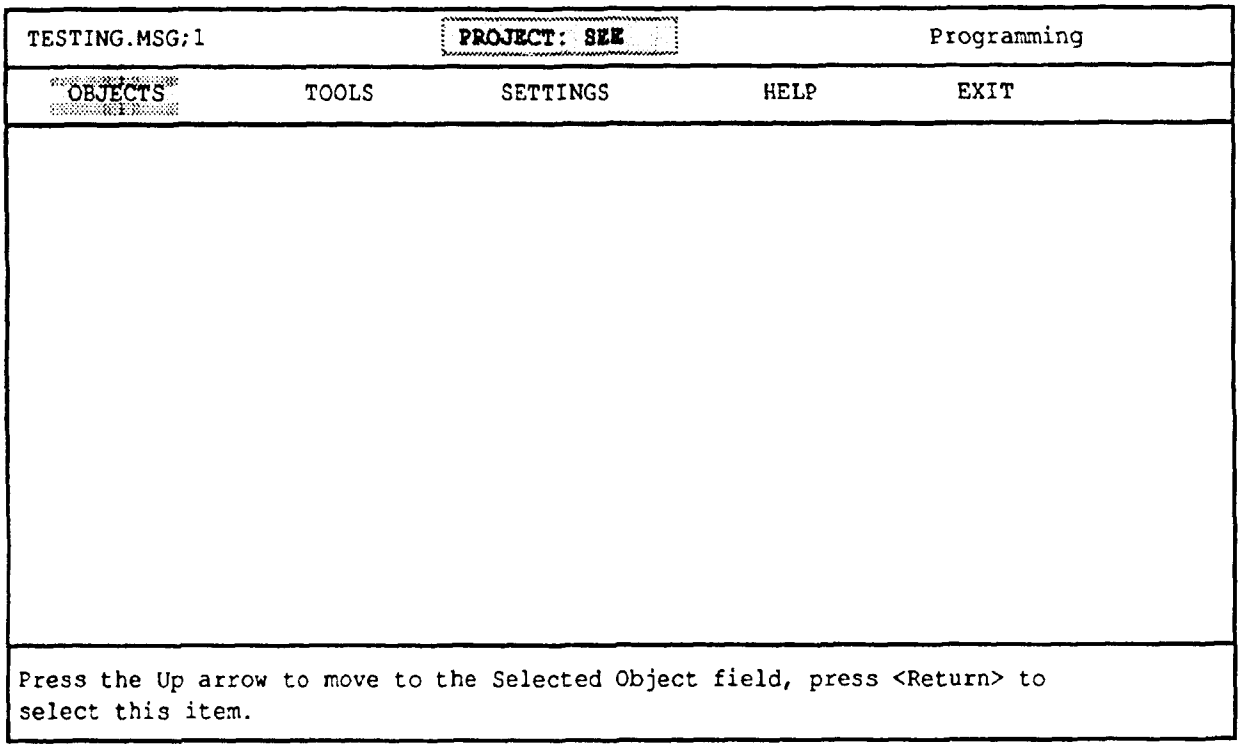


Figure A-1. Top-level screen of SLCSE.

Figure A-2 shows the top-level SLCSE screen with the Objects Menu pulled down. Figure A-3 shows the Tools Menu pulled down. These menus are included to assist the user when referring to the basic operations provided below.

A.1 BASIC OPERATIONS

A.1.1 Accessing Tools

1. Press right or left arrow until TOOLS in the menu bar is highlighted. <return>.

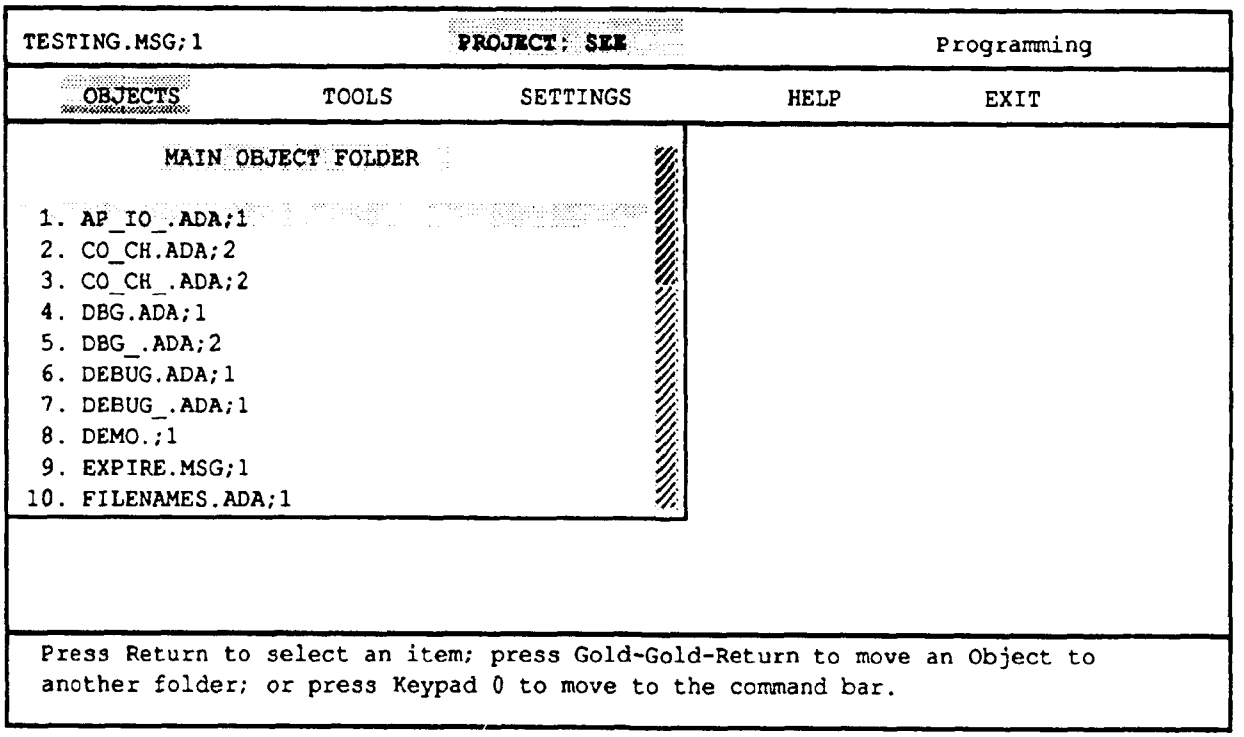


Figure A-2. Top-level screen of SLCSE with Objects Menu pulled down.

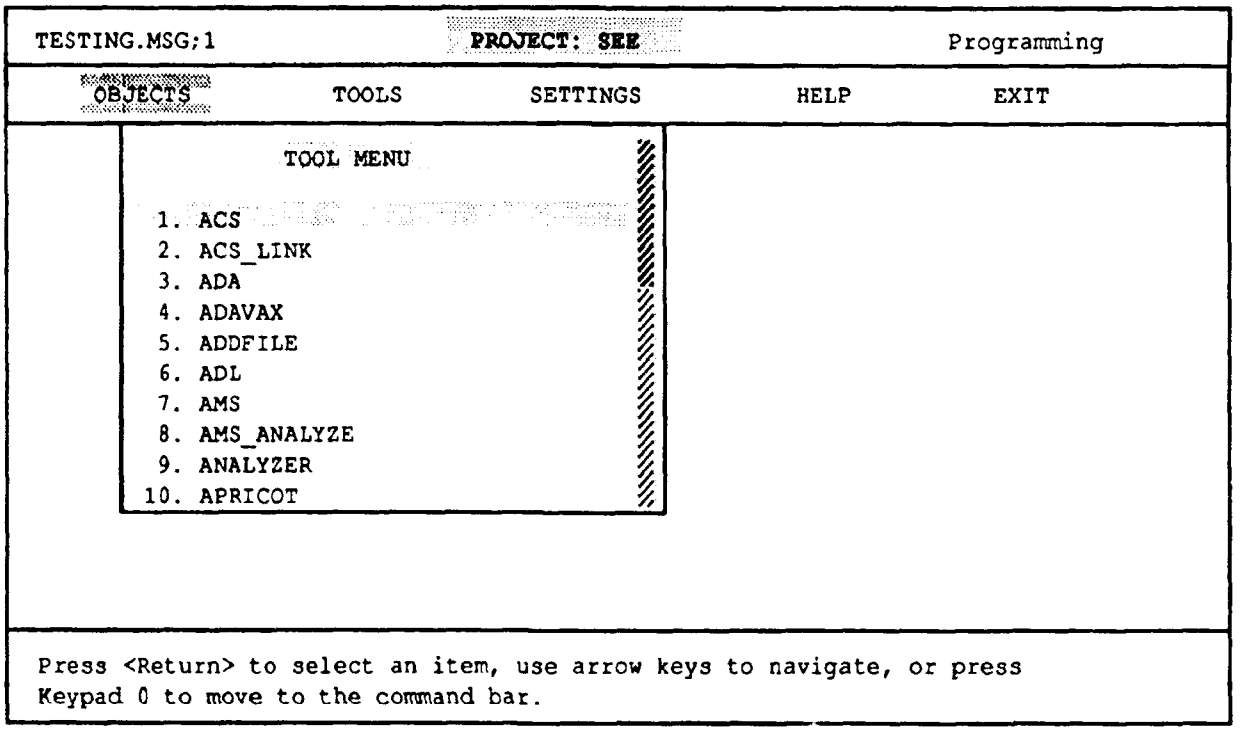


Figure A-3. Top-level screen of SLCSE with Tools Menu pulled down.

2. In the Tool Menu arrow to desired tool (by pressing up and/or down arrow.)
3. If tool requires a filename, other parameter, or a qualifier; press PF1 (Gold Key) and <return>. Otherwise <return>.
4. Read user's manual for tool being used to discover what the qualifiers mean, what entries to use, and to answer any questions.

A.1.2 Importing Files Into SLCSE

Before using most SLCSE tools, the tool input file must first appear in the SLCSE's OBJECT list. For the file to appear requires that it either be created in the SLCSE (by copying or editing a file, or compiling, linking, or executing a tool) or be imported into the SLCSE from elsewhere on the VMS system. The steps to import a file are

1. Arrow to TOOLS on the menu bar. <return>
2. Select IMPORT from the tool list.
3. Press PF1. <return>
4. <return> (on SETUP on the menu bar).
5. Type in the name (including the directory path) of the file to be imported.
6. <return>
7. Type in the name the file is to have in the SLCSE.
8. Press keypad 0.
9. <return> (on INVOKE in the menu bar).
10. Arrow to DONE on the menu bar. <return>

A.1.3 Moving Directly to a Given Tool

To move directly to a tool in the tools menu, enter the tool number, (shown to the left of the tool name), using numbers across the top of keyboard. (See figure A-3.)

A.1.4 Moving from an Object in the Objects Menu to the Tools Menu

This operation is done repeatedly when using the SLCSE. The normal method of executing a tool is to first select the file from the Objects Menu that is to be input to the tool.

Press PF1 right arrow.

A.1.5 Moving from a Tool in the Tools Menu to the Objects Menu

Press PF1 (Gold Key) left arrow.

A.1.6 Scrolling Menus

To scroll in the Object Menu, Tool Menu, and Settings Menu, press keypad 8. Press keypad 4 (forward) or 5 (backward) to change scrolling directions.

A.1.7 Selecting an Object from the Objects Menu

Move up or down the the Objects Menu by pressing the up or down arrows. Once the desired object is highlighted, <return>.

A.1.8 Submitting Batch Jobs

Some tools allow jobs to be submitted to the batch queue. Do this in the Tool Setup Window by toggling (press keypad 1) from INTERACTIVE to BATCH.

A completion message is generated by the SLCSE when the batch job completes. No audible signal is generated.

A.1.9 Viewing Messages Generated by Tool

If executed interactively, press keypad 3.

If executed in batch mode, press PF1 (Gold Key) keypad 3.

A.1.10 VMS-like Tools

The following tools perform the same operation as the VMS commands of the same name. These tools use a Setup window.

1. COPY
2. DELETE - see explanation below.
3. DIRECTORY
4. EDT - the VMS editor
5. MAIL
6. PURGE - see explanation below.
7. RENAME
8. TYPE

A.1.10.1 Deleting Files. To delete a specific file, select the file from the OBJECTS menu, then select DELETE from the TOOLS menu.

To delete more than one file, press PF1 and <return> on the DELETE tool in the TOOLS Menu, then, if necessary, use wild cards in the file name. The user can be prompted before each file is deleted to make sure it is alright to delete the file. This inquiry can be disabled if the user chooses.

A.1.10.2 Purging Files. To purge a specific file, type the filename into the purge window displayed when the PURGE tool is selected. To purge more than one filename, or the entire directory, use wild cards in the file name. The PURGE tool deletes all but "Purge Version Limit" copies of files. "Purge Version Limit" is set using the SETTINGS menu.

A.2 SLCSE MANAGER OPERATIONS

All of the following actions require access to the SLCSE manager account. Logging into that account will be the unmentioned first step for each action.

A.2.1 Defining Roles

The SLCSE requires tools be assigned to the user roles (e.g., system analysis, project management). The process is called defining the roles. Roles are defined by using the SLCSE Environment Manager and following the steps given below.

1. Type: sem. <return>. (This will bring up the screen shown in figure A-4.)

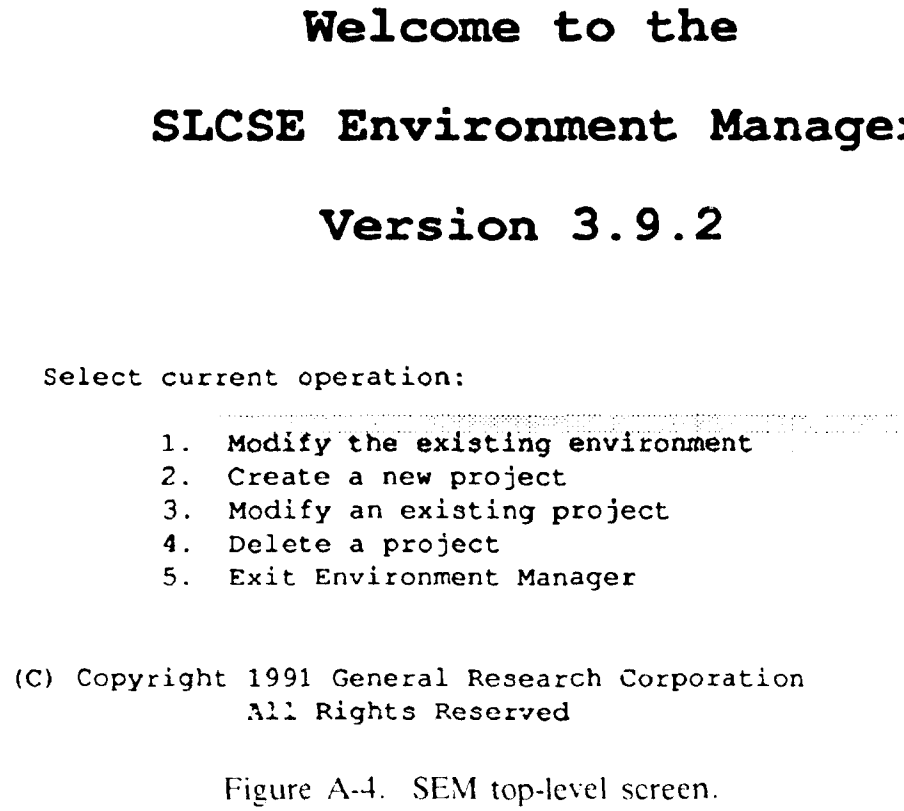


Figure A-4. SEM top-level screen.

2. Highlight choice "1. Modify Existing Environment." <return> and <return>

3. In the menu bar, arrow to ROLES. <return> (This will bring up the screen shown in figure A-5.)

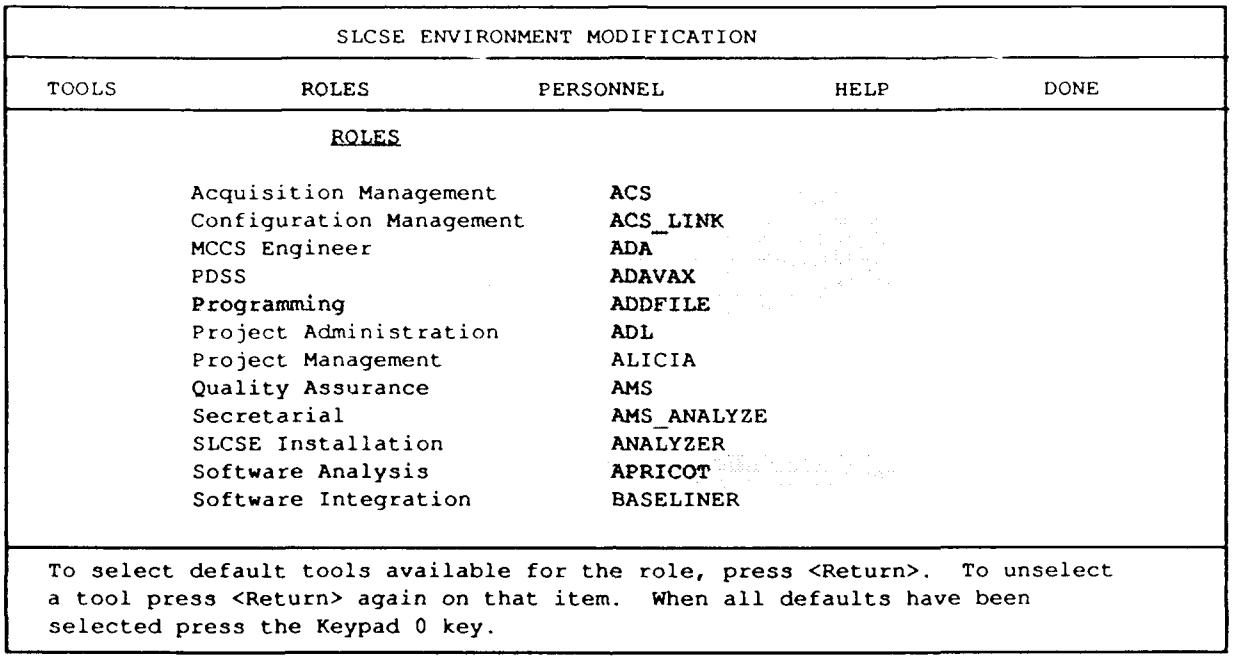


Figure A-5. SEM when assigning tools to roles.

4. Arrow to desired role. <return>

5. Arrow to a tool required for the role. <return>

6. Repeat last step until required tools are highlighted.

7. If any tool is highlighted that is not required, arrow to it. <return>

8. Repeat last step until only required tools are highlighted.

9. Type: Keypad 0.

10. Repeat steps 4 - 9 until all required roles have been defined.

11. Type: Keypad 0.

12. Arrow to DONE. <return>, <return>, and <return>

13. Arrow to choice "5. Exit..." <return>

A.2.2 Adding Personnel to Environment

Before a person can be assigned to a project under SLCSE, they must first be added to the personnel list under the environment. When a person is added to the environment, SLCSE will verify their user name exists on the system before creating the files internally required by the SLCSE. Users are added by using the SLCSE Environment Manager and following the steps given below.

1. Type: sem (This will display the screen shown in Figure A-4.)
2. Highlight choice "1. Modify Existing Environment". <return> and <return>.
3. Arrow to PERSONNEL in the menu bar. <return> (See figure A-6.)

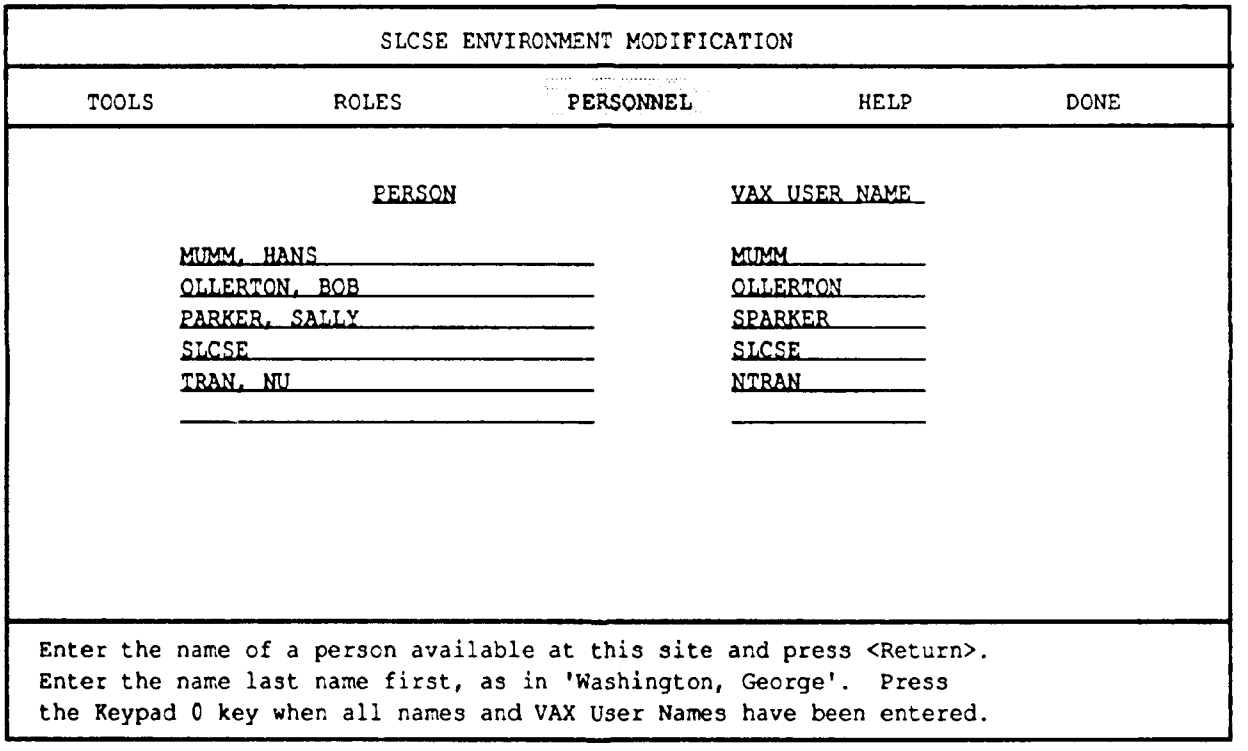


Figure A-6. SEM screen for adding personnel to the SLCSE environment.

4. Type the person's name (last name first) on the highlighted blank line in the first column. Type their user name in the second column. (See figure A-6.)
5. Type: Keypad 0.
6. Arrow to DONE. <return>, <return> and <return>.
7. Arrow to choice "5. Exit..." <return>

A.2.3 Creating a Project

Before any work can be done on a project using the SLCSE, the project must first be created in SLCSE. That is, the SLCSE must be notified of the project's existence, the project's users, the database to be used, the roles the users can take, and the tools that each user is allowed to use. To create a project follow the steps shown below.

1. The SLCSE manager must obtain special system privileges.

   Type: set process/priv=all

2. Run the Database Administrator to create the project database.

   Type: dba <return>

   (Results in the screen shown in figure A-7.)

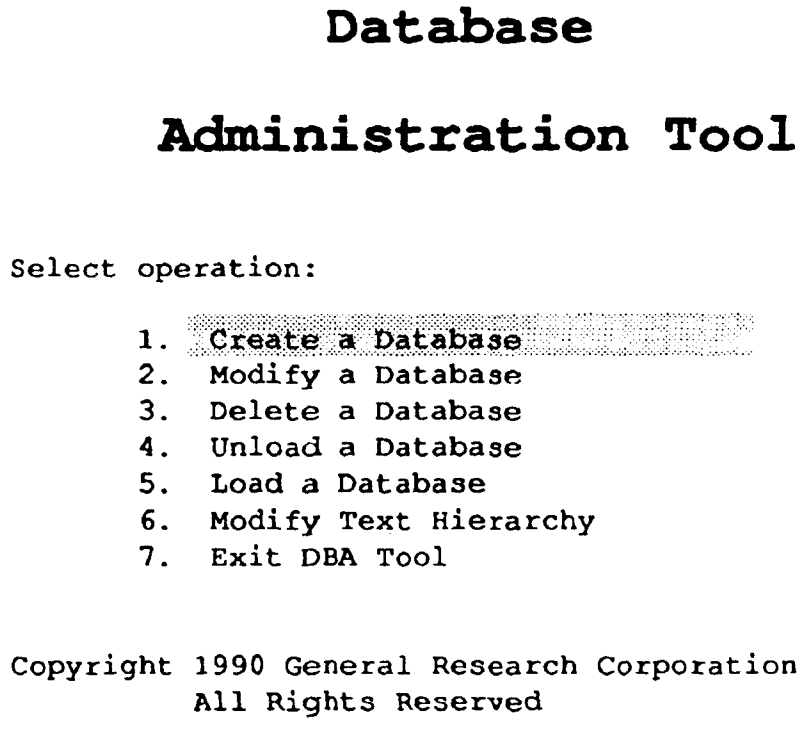


Figure A-7. First DBA tool.

3. Complete database administration information. (See figures A-8 through A-11.)

   a. Arrow to choice "1. Create a Database." <return> (Results in screen shown in figure A-8.)

Create Database Checklist

Mandatory operations:

_____ 1. Define Database
_____ 2. Compile Schema
_____ 3. Create Database

Optional operations:

_____ 4. Load DGL Data Files
_____ 5. Load Narrative Text
_____ 6. Create Metaschema Tables
7. Exit Checklist Menu

This option allows you to define the database name, disk, type, and number of text attribute hierarchies. This step must be completed before other options are selected.

Figure A-8. First database creation screen.

Define Database

Database Name: _____ Database Disk: _____

Number of Text Attribute Hierarchies: 1 Database Type: RDB

Text 1: _____

CANCEL INVOKE

Figure A-9. Database definition screen.

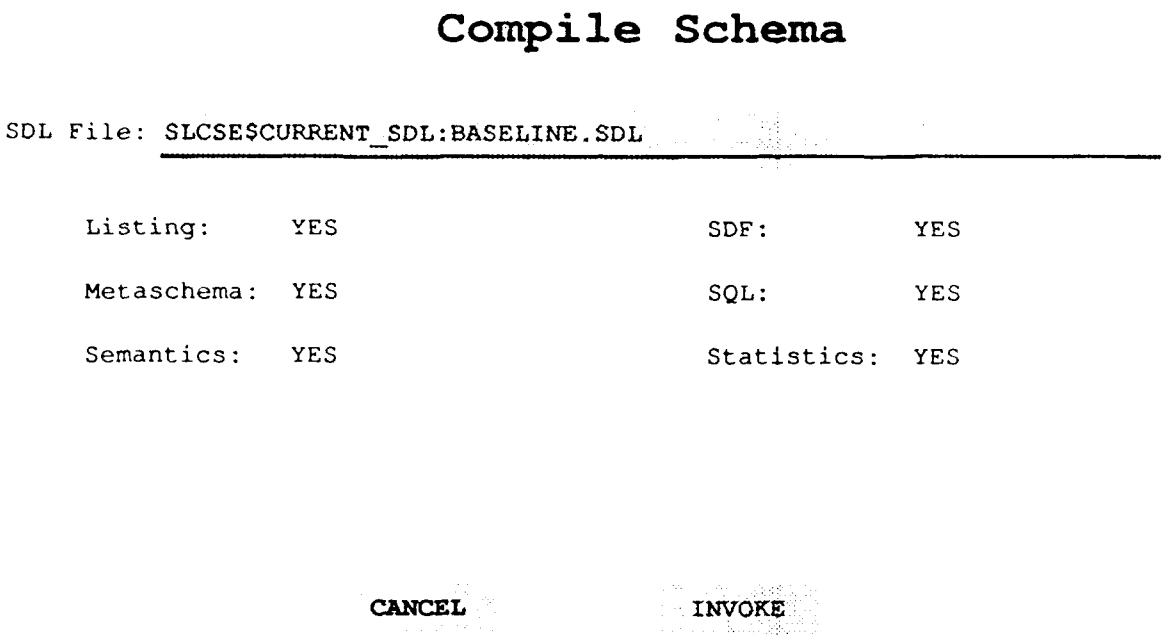


Figure A-10. Creating a schema DBA screen.

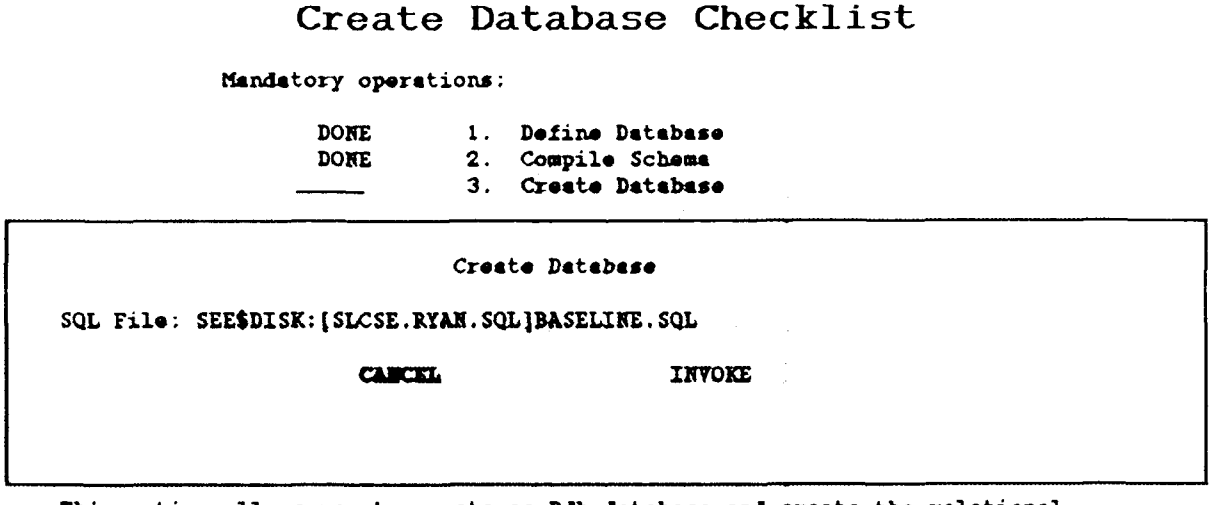


Figure A-11. Final screen of mandatory database creation steps.

b. Arrow to choice "1. Define Database." <return> (Results in screen shown in figure A-9.) Fill in the screen following the instructions below.

Field Explanation of field

1. Enter name for database. <return>
2. Enter name of disk where database will reside. <return>
3. Type: 1 (for the number of text attribute hierarchies). <return>
4. Leave database type as RDB. (Currently the DBA Tool only supports the creation of RDB databases.)
5. Enter name of disk (for text 1). Usually matches the one given in step 2 as disk where database resides. <return>

   Arrow to Invoke. <return>
   (Results in a batch job submission.)

c. After the batch job has completed, arrow to choice "2. Compile Schema." (See figure A-8.) <return>
(Results in screen shown in figure A-10.)

1. <return>
2. For Metaschema, use Keypad 1 to toggle to "NO" (unless the ALICIA tool will be used on the project). All other fields should keep their default values.
3. Arrow to Invoke. <return>.
   (This results in a batch job submission.)

d. After the batch job has completed arrow to choice "3. Create Database." (See figure A-8.) <return>
(Results in screen shown in figure A-11.)

1. <return>
2. Arrow to Invoke. <return>

e. Arrow to choice "7. Exit Checklist Menu." (See figure A-8.) <return>

f. Arrow to choice "7. Exit DBA Tool." (See figure A-7.) <return>

4. Run the SLCSE Environment Manager.

a. Type: sem (See figure A-4.)

b. Highlight choice "2. Create Project." <return>

c. Enter name of project. <return>

d. Arrow to NETWORK. <return>

e. Complete network form. (See figure A-12.)

1. Enter name of database used when working with dba tool above.
2. Enter name of disk used when working with dba tool above.
3. Enter device and directory path of the SLCSE CM directory.
4. Enter device and directory path of the SLCSE SDF directory.
5. Type: 2 (for SDF purge limit)
6. Type: Keypad 0

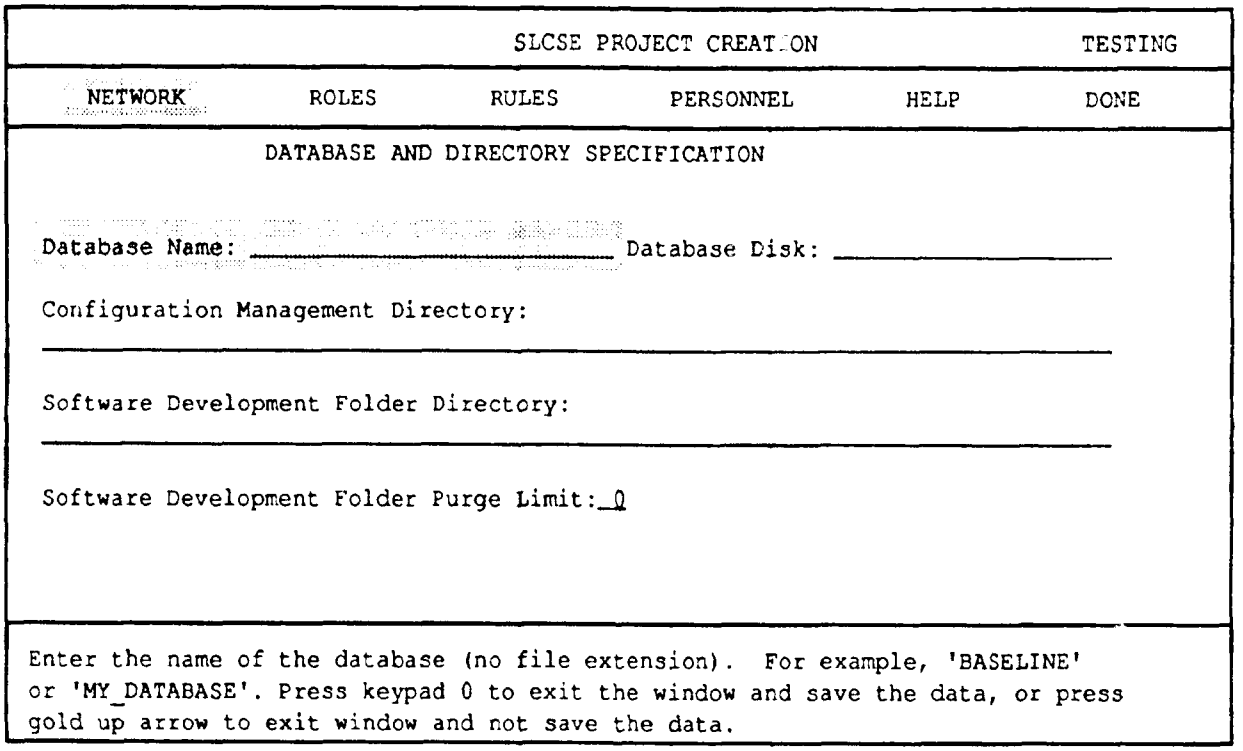


Figure A-12. SEMs network definition form.

f. Define roles.

1. Arrow to ROLES. <return>
2. Arrow to desired role. <return>
3. Arrow to a tool required for the role. <return>
4. Repeat last step until are required tools are highlighted.
5. If any tools are highlighted that are not required, arrow to it. <return>

6. Repeat last step until only required tools are highlighted.
7. Type: Keypad 0.
8. Repeat from "Arrow to desire role" until all required roles have been defined.
9. Type: Keypad 0.

g. Assign personnel to project following steps as given in section A.2.4 below.

h. Arrow to DONE. <return>, <return> and <return>.

i. Arrow to choice "5. Exit..." <return>

A.2.4 Adding Personnel to a Project

Personnel required for a project must be assigned to that project within the SLCSE. First, make sure the required personnel have been added to the SLCSE (see section A.2.2), then use the SLCSE Environment Manager to add them to the project.

1. Type: sem
2. Arrow to choice "3 Modify an Existing Project." <return>
   (See figure A-4.)
3. Arrow to the desired project. <return>
4. Arrow to PERSONNEL in the menu bar. <return>
5. Arrow to desired user name. Press PF1 and <return>

(A user name may be unselected by pressing <return> again.)

6. Arrow to role this person will have on project . <return>
7. Repeat last step for all roles this person will hold.
8. Type: Keypad 0 twice.
9. Arrow to DONE. <return>, <return> and <return>.
10. Arrow to choice "5. Exit Environment Manager" <return>
    (See figure A-4.)

A.2.5 Modifying the Toolset Available to a User Role

The SLCSE may be tailored at the project level by changing the collection of tools that a user role may access. This type of tailoring requires the use of the SLCSE Environment Manager.

1. Type: sem. <return>
2. Arrow to choice "3. Modify an Existing Project." <return>
   (See figure A-4.)
3. Arrow to project to be tailored. <return>
4. Arrow to ROLES in the menu bar. <return>
   (During this process the screen will resemble figure A-13.)

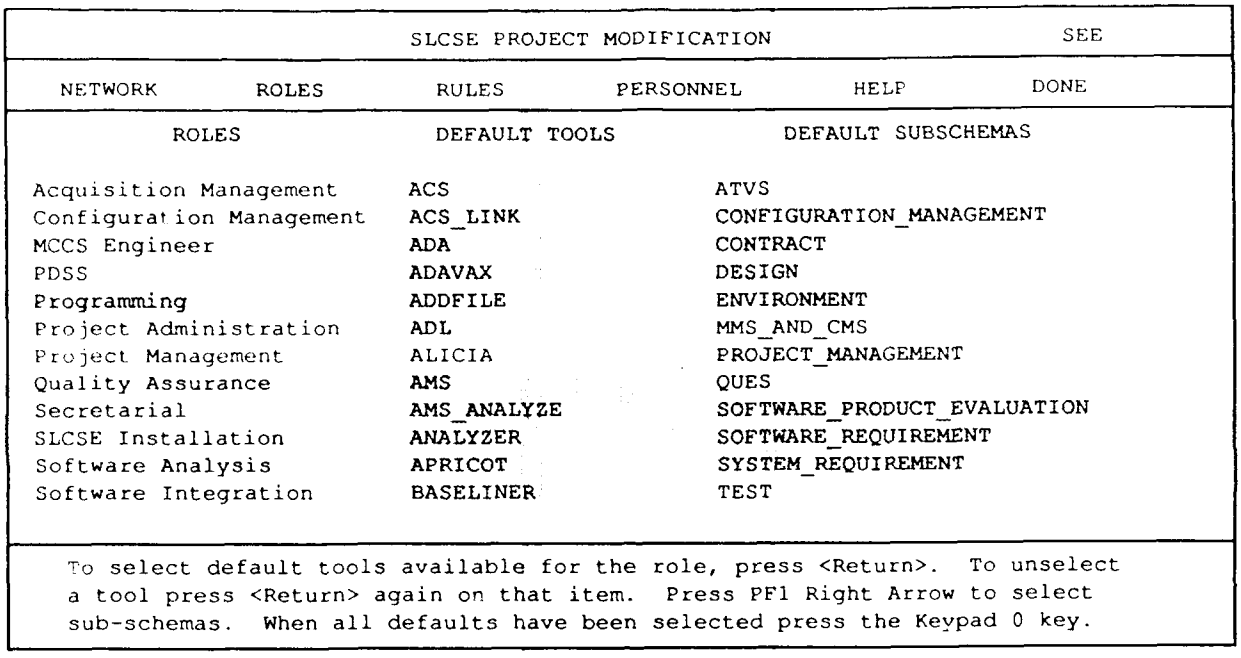


Figure A-13. Modifying roles' default tools for a project.

5. Add tools as described in Defining Roles (see section A.2.1).
6. Delete default tools as described in Defining Roles.
7. Type: Keypad 0 twice.
8. Arrow to Done, <return>, <return>, and <return>
9. Arrow to choice "5. Exit Environment Manager" <return>

A.2.6 Modifying the Toolset Available to a User

If a specific user needs access to a tool not allowed within the user's role follow the steps below. Also, follow them when a specific user is not to have access to a tool that is allowed for their role.

1. Type: sem. <return>
2. Arrow to choice "3. Modify an Existing Project." <return>
   (See figure A-4.)
3. Arrow to project to be tailored. <return>
4. Arrow to PERSONNEL in the menu bar. <return>
5. Arrow to person's name. Press PF1 and <return>
6. Arrow to the role to be modified and press PF1 and <return>
7. Arrow to the tool to be added or deleted. Highlight or un-highlight by pressing <return>. (Highlighted means the user will have access to it.)
8. Repeat last step, until person's role tool set is correct.
9. Type: Keypad 0 three times.
10. Arrow to DONE. <return>, <return> and <return>.
11. Arrow to choice "5. Exit Environment Manager" <return>

A.3 OTHER OPERATIONS

A.3.1 Making Hard Copies of Screens

To generate copies of the screens from a PC, use the PC as VT100 and do Control-Print Screen. To print the extra characters displayed on the screen (other than the ASCII character set) to a laser printer the symbol set for these characters must be used. Refer to your printer manual for instructions on how to select the symbol set.

To generate copies of the screens from a Mac, use the Mac as a VT100 and use the Screen Selection function on the File menu of most terminal programs. NOTE: This may not generate the graphics portion of the screens correctly.

APPENDIX B: DETAILED STEPS FOR TOOL INTEGRATION

B.1 INTEGRATING SIMPLE TOOLS

A simple tool is any tool that has no qualifiers and either takes no parameters or prompts the user for the parameters. The major steps in this section follow the process shown in figure 4-1. The titles of the rectangles in figure 4-1 (for integrating non-UI-conformant tools) correspond to the titles of this section (B.1.1 through B.1.5). The user must first login to the SLCSE manager's account. Throughout these instructions "tool_call" is used as a place holder for the name of the tool being integrated. For example, if the tool JUMBO is being integrated, then everywhere the instructions say "tool_call" enter "JUMBO."

B.1.1 Define Tool Symbol

The tool symbol can be defined in the SLCSE setup command file (slcse_setup.com), the SLCSE startup command file (sys$manager:slcse_startup.com), or in the system login command file (sys$startup:sylogin). If the tool symbol is defined in the sylogin.com, then users can use the tool symbol to run the tool whether or not they are using the SLCSE, even if the user doesn't have access to the SLCSE. If the tool symbol is defined in slcse_setup.com or slcse_startup.com, it can only be used to run the tool by users with access to the SLCSE.

To define the tool symbol in the file slcse_setup.com, follow the steps given below.

1. Type: Edit [slcse]slcse_setup.com
2. Add line: $ Tool_call :== run [directory_path]tool
3. Exit and save file

Editing the sys$manager:slcse_startup.com file requires system privileges and should probably be done by the System Administrator. If the System Administrator grants the user system privileges to define the tool symbol in the file sys$manager:slcse_startup.com, follow the steps given below.

1. Type: Edit sys$manager:slcse_startup.com
2. Add line: $ Tool_call :== run [directory_path]tool
3. Exit and save file

Editing the sys$startup:sylogin.com file requires system privileges and should probably be done by the System Administrator. If the System Administrator grants the user system privileges to define the tool symbol in the file sys$startup:sylogin.com, follow the steps given below.

1. Type: Edit sys$startup:sylogin.com
2. Add line: $ Tool_call :== run [directory_path]tool
3. Exit and save file

The SLCSE Environment Manager is used in steps B.1.2 through B.1.5.

B.1.2 Enter Tool Name In SEM

This entering is accomplished by following the steps listed below.

1. At prompt, type: sem
2. Highlight selection "1. Modify the Existing Environment," <return>. (See figure A-4.)
3. Return on environment name
4. Highlight "TOOLS" in menu bar, <return>. (See figure A-5.)
5. On blank line type: tool_call. (See figure B-1.)

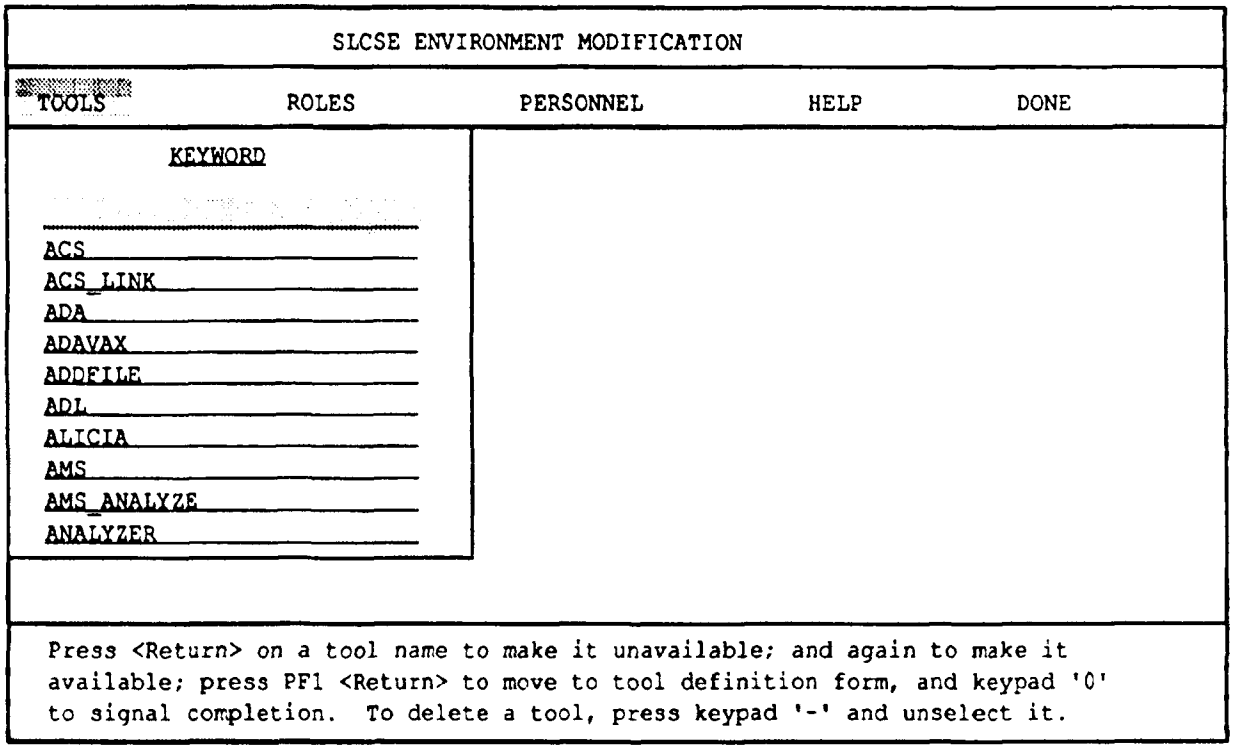


Figure B-1. Screen as it appears before adding a tool.

NOTE: Tool_call in this step should match the one defined as the tool symbol in step B.2.1.

6. Press PF1 and <return>
   This will bring up the tool definition form.

B.1.3 Define Tool Parameters

The tool parameters are defined by completing the tool invocation data form shown in figure B-2. Below is a list of the fields in this form and an explanation of how to complete each field. The numbers in figure B-2 correspond with the numbers used in the instructions below.

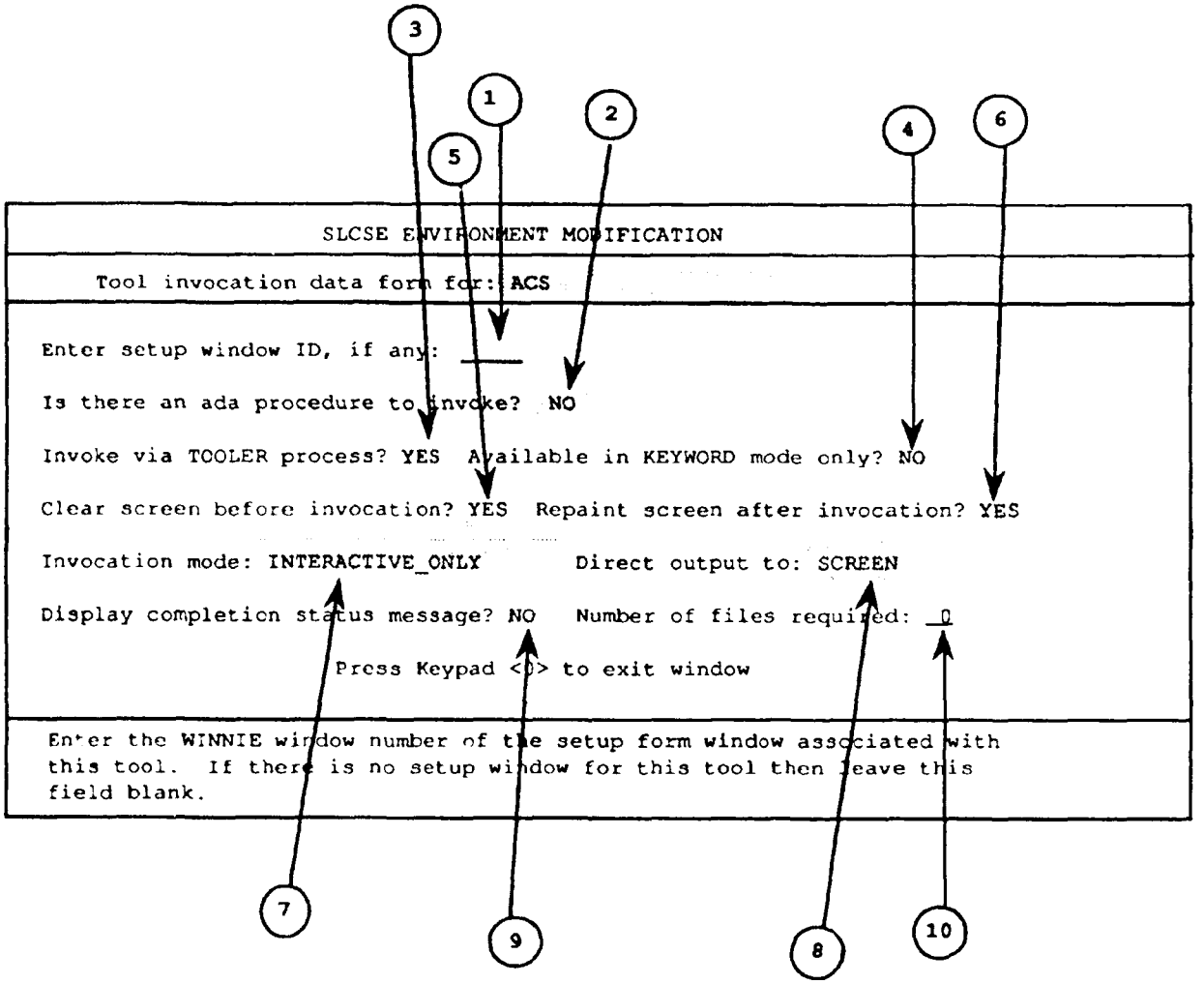


Figure B-2. Tool invocation data form for simple tool.

1. <return>
   (A simple tool has no setup window and the first field is blank.)
2. Field should be "NO," toggle (Keypad 1) if necessary. <return>
   (A simple tool never requires an Ada procedure.)
3. Field should be "YES," toggle if necessary. <return>
4. Field should be "NO," toggle if necessary. <return>
5. If tool will produce output to screen, toggle to "YES," otherwise toggle to "NO." <return>
6. If last field was "YES," then toggle to "YES," otherwise toggle to "NO." <return>
7. If tool will require user interaction before running, toggle to "INTERACTIVE_ONLY." If tool will always be run as a batch job, toggle to "BATCH_ONLY." Otherwise, toggle to "BATCH_OR_INTERACTIVE." <return>
8. If it's possible to use the tool interactively, toggle to "SCREEN," otherwise toggle to "OUTPUT_FILE." <return>
9. If it's possible to use the tool interactively, toggle to "NO," otherwise toggle to "YES." <return>
10. Should show a "0." If not enter a "0." <return>
11. Type: Keypad 0.

The form shown in figure B-2 is completed for a simple tool that produces output to the screen and can only be used interactively.

B.1.4 Associate Tool With Role(s)

Refer to figure A-5 when performing the steps given below.

1. Highlight "ROLES" in the menu bar. <return>
2. Select role that will use Tool_call. Highlight chosen role. <return>
3. Highlight Tool_call in the default tools list. <return>
4. Type: Keypad 0.
5. If more than one role will use Tool_call, repeat the previous three actions.
6. Type: Keypad 0.
7. Highlight "DONE." <return>

8. <return> in response to both questions.

B.1.5 Modify Projects

1. Highlight "3. Modify existing projects." <return>
   (See figure A-4.)
2. Highlight project to be modified. <return> and <return>
3. Modify personnel by doing the actions listed below.
   a. Highlight "PERSONNEL" in the menu bar. <return>
   b. Highlight person who will use Tool_call. Type: PF1 and <return>
   c. Highlight role the person will be using Tool_call in. Type: PF1 and <return>
   d. Verify that Tool_call is in reverse video. If it's not, highlight it. <return>
   e. Type: Keypad 0 twice.
   f. If more than one person on project will be using the tool, repeat actions B through E.
   g. Type: Keypad 0.
   h. Highlight "DONE." <return>
   i. Type "Y" in response to both questions.
4. Highlight "5. Exit the Environment Manager." <return>

B.2 INTEGRATING TOOLS WITH QUALIFIERS

Throughout these instructions "tool_call" is used as a place holder for the name of the tool being integrated. To integrate tools the user must be logged into the SLCSE Manager's Account. The steps in this section follow the order shown in figure 4-1.

B.2.1 Define Tool Symbol

A DCL command must be created and included in the DCL table for tools with qualifiers. These actions require system privileges and should be taken only by the System Administrator.

B.2.2 Define A Setup Window

This section describes the steps to follow to define setup windows. The BBoard is used as an example.

B.2.2.1 Determine Tool Parameters and Qualifiers. The user determines the tool parameters and qualifiers by examining the tool user's manual. In a well-written user's manual this information is clearly stated.

B.2.2.2 Design the Window Layout. The following should be considered

1. Can tool be used only as a batch job, only interactively, or both?
2. If the tool uses a file name input, and the request for the file name is the first fill-in field on the screen, the user can choose to update it with the selected object, otherwise the user can not.
3. Should fields be fill-in or toggle fields? Toggle usually works best with qualifiers and fill-in best with parameters.
4. Do the qualifiers require parameters? For example, if EXPIRE is a qualifier it usually requires a date be given as a parameter.

B.2.2.3 Define WINNIE Commands. In this section, first some relevant WINNIE definitions will be given. Then an explanation of the BBoard WINNIE commands is provided. The BBoard represents a typical tool with qualifiers that a user might wish to integrate into the SLCSE. Finally, the steps for constructing the WINNIE commands will be explained.

The WINNIE definitions for all SLCSE Command Executive windows are defined in the file SLCSESUI:CE_WIN.ASC. These definitions are in window number order in this file. The window definitions are free format, i.e., there may be a varying number of spaces between data items.

B.2.2.3.1 WINNIE Definitions. The definition of terms is provided before going into a detailed explanation.

Field - A field is a piece of information contained in a setup window. A field is needed for executing a tool. WINNIE supports fill-in fields and toggle fields. A fill-in field is one where the user enters characters from the keyboard. A toggle field is one where the user makes a selection from a default set of values.

Field Number - This is an integer number that uniquely defines a field within a window.

Invisible Field - This is a field that is not initially displayed in the setup window upon tool invocation. This field is not displayed until the MOO commands change its status to visible.

Protected Field - In WINNIE all fields that can be toggled must be protected. A protected field is one whose text value can not be edited by the user.

Toggle Number - A number indicating the order (i.e., which element in an enumeration set) in which default text will be displayed.

Visible Field - This is a field that is initially displayed in the setup window upon tool invocation.

Window - Rectangular regions on a monitor screen. They may have a frame (border). The first screen shown in figure 4-2, for example, consists of four windows. These windows are the title window (top), the menu bar (below the title window), the setup window, and the prompt window (bottom).

Window Identification Number - A number that uniquely identifies each window in the SLCSE. Numbers may range from 1 to 400. These window ID numbers are also used by MOO.

B.2.2.3.2 BBoard Example. Figure B-3 contains the WINNIE commands required to construct the BBoard setup window (figure 4-2). Figure B-3 will be referred to repeatedly in this section. The WINNIE commands the user must change when creating a setup window will be explained in detail. Those commands that typically do not change will only be discussed at high level. The user who wishes to learn more about WINNIE than is provided here should refer to (Cooper, 1986). Additional information that a user may need for setup window, but that is not covered by this example, is also provided.

The WINNIE BBoard commands given in figure B-3 will be explained and referred to using the circled numbers. The numbers below correspond to the circled numbers.

(1) Each window definition begins with the window statement. This statement indicates that the BBoard setup window number is 324. This window begins in column 1 and extends to column 80. The bottom row of the window is 21 and the top row is 5.

(2) These four commands stay the same for all tool setup windows. They define the frame around the window, the location of the scroll bar, the layout of the keypad, and give the form number.

(3) This prompt statement occurs before any field statements; therefore, the fields defined after this default prompt statement will inherit this prompt. The prompt may be redefined for a particular field by defining another prompt within the field definition. These commands define the two-line default prompt appearing at the bottom of the setup window (figure 4-2). The first line of the prompt appears in Window 3 beginning in line 1. The second line of the prompt begins in Window 3, line 2.

(4) These three lines define the characteristics of Field 35. SLCSE Field 35 is generally used for interactive or batch job submission. The first number on the field statement is the field number, followed by the line and column of the field starting position, and the field width. This is followed by the video

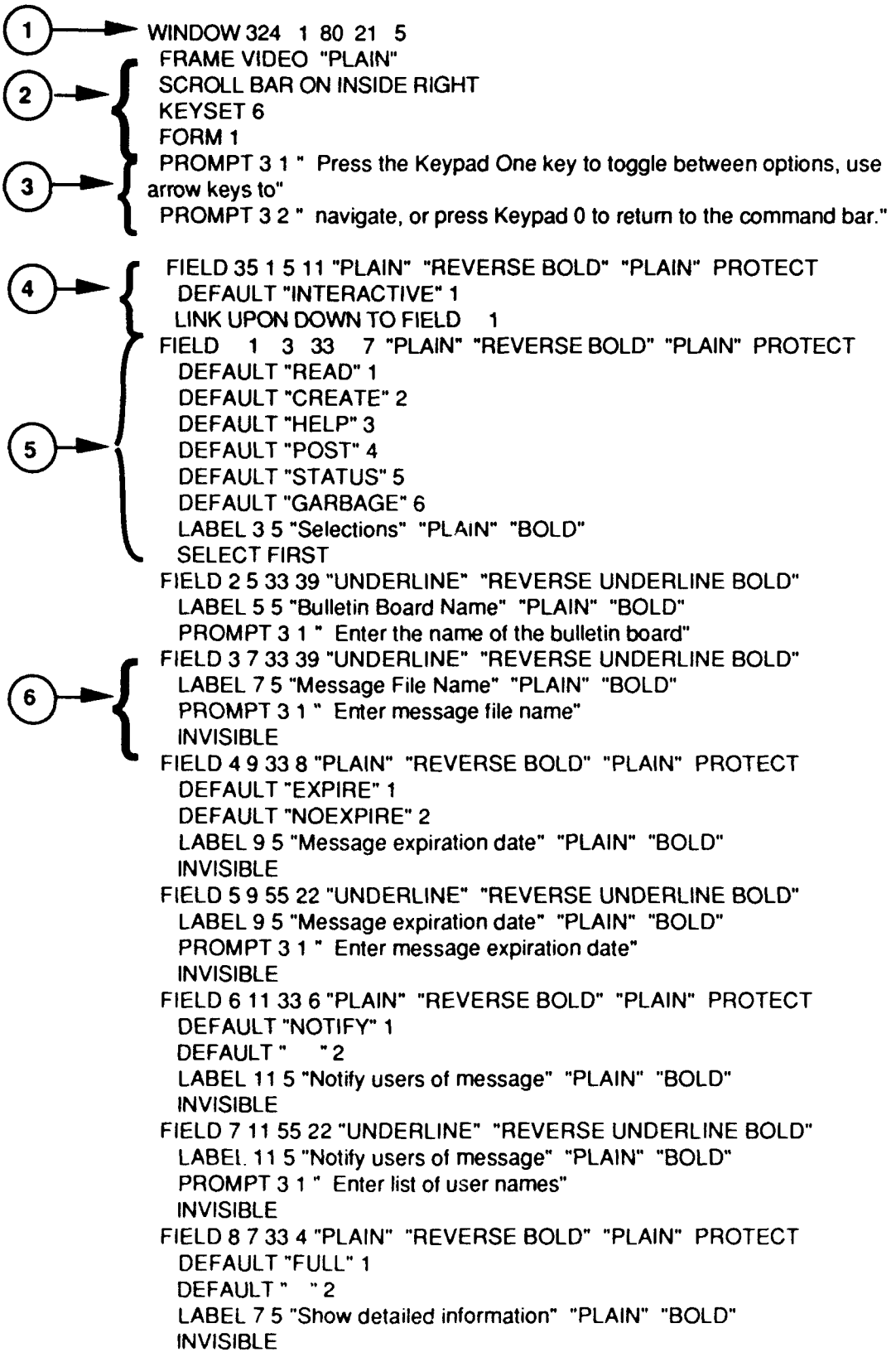


Figure B-3. BBoard setup window definitions using WINNIE.

```
FIELD     9   9 33   6  "PLAIN" "REVERSE BOLD" "PLAIN"  PROTECT
  DEFAULT   "SCREEN"        1
  DEFAULT   "OUTPUT"2
  LABEL      9   5  "Direct output to "  "PLAIN"  "BOLD"
  INVISIBLE
FIELD    10   9 55  22  "UNDERLINE" "REVERSE UNDERLINE BOLD"
  PROMPT     3   1  " Enter output file name"
  LABEL      9   5  "Direct output to "  "PLAIN"  "BOLD"
  INVISIBLE
FIELD    11   7 33   3  "PLAIN" "REVERSE BOLD" "PLAIN"  PROTECT
  DEFAULT   "LOG"1
  DEFAULT   "  "    2
  LABEL      7   5  "Show detailed information"  "PLAIN"  "BOLD"
  INVISIBLE
FIELD    12   9 33   5  "PLAIN" "REVERSE BOLD' "PLAIN"  PROTECT
  DEFAULT   "PURGE" 1
  DEFAULT   "  "        2
  LABEL      9   5  "Direct output to "  "PLAIN"  "BOLD"
  INVISIBLE
```

Figure B-3. BBoard setup window definitions using WINNIE (continued).

characteristic of how the field should look when the cursor is not on the field, the video characteristic of the field when the cursor is on the field, the video characteristic of the field after it has been selected (user pressed return and cursor has moved off field), and PROTECT which means the field can not be changed by the user.

The second line is the field default statement. It contains the default text which is displayed in the setup window and the toggle number. The BBoard may only be run interactively, thus there is only one field default statement.

The third line instructs the WINNIE program to move the cursor to the Field 1 after the user presses return. In this example this command is not needed. If the "UPDATE WITH SELECTED OBJECT" field was present in this window and located on the same line of the window as Field 35, then WINNIE would move the cursor to it by default. The "LINK UPON DOWN TO FIELD 1" statement instructs WINNIE to move the cursor to Field 1 after the user presses return.

WINNIE, by default, positions the cursor on the top left field in the window. When the user presses return, WINNIE moves the cursor to the next field on the same line (to the right). If there are no fields on the same line, WINNIE moves the cursor down to the next field.

(5) These nine lines define the characteristics of Field 1. It begins in line 3, column 33 and has a width of 7 characters. When the cursor is not on the field it will be displayed "PLAIN" video. When the cursor is on the field it will be in "REVERSE BOLD." Then when selected it will be "PLAIN" again. It is protected.

The six field-default statements define the text that is displayed when the user toggles through the BBoard options.

The next line is the field-label statement. It defines a label for this field. The label "selections" is defined to appear in line 3, column 5. When the cursor is not positioned on the field, it will be displayed in "PLAIN" video. When the cursor is positioned on the field, it will appear in "BOLD."

The select-first statement tells the cursor to be positioned on this field when the window is first entered.

(6) These four lines define the characteristics of Field 3. Only the fourth line, the invisible field statement, has not been explained. This statement says that the field will not be visible when the BBoard window is initially displayed. The window will become visible when MOO commands change its status. The remaining fields in figure B-3 are filled in a similar manner.

Field 500 is frequently used to define SLCSE setup windows, but that did not appear in the BBoard example. This field is used to automatically insert the selected object name in the setup window.

EXAMPLE:

```
FIELD 500 1 43 33 "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
 DEFAULT "UPDATE WITH SELECTED OBJECT" 1
 DEFAULT "DON'T UPDATE WITH SELECTED OBJECT" 2
```

If the field is set to the first default option, then the SLCSE uses the file(s) (the user can select more than one file) selected by the user to automatically complete the first fill-in field. If the user toggles the field contents to the second default option, then the user must complete the fill-in field manually.

B.2.2.3.2 Building WINNIE Commands In General.

1. Determine the window identification number for the tool. Find a window number that does not currently exist in the CE_WIN.ASC file, and use it.
2. Copy an existing window that is similar to the desired window for the new tool.
3. Paste the copy into the proper place in the file, windows are in numerical order. (Numerical order is used for readability; WINNIE does not require it.)

4. Change window number to the new one.

5. If the tool can only be used in only batch or only in interactive mode, change Field 35 to reflect this.

6. If tool can not use selected object as first fill-in parameter, change Field 500 to reflect this.

7. Actual creation of the fields for the window is tool dependent. However, there are some guidelines. (Refer to figures B-3 and C-1 through C-4.)

   - With the exception of Fields 35 and 500, fields are usually in numerical order within a window definition.
   - Labels are positioned to the left of the corresponding field in the window.
   - In toggle fields, all possible choices must be labeled as DEFAULT and followed by a number. The choices will be toggled through in numerical order.
   - In toggle fields, on the line that begins FIELD the last word should always be PROTECT.
   - It is easier to write the required Ada code later, if the toggle values are actual qualifier values. For example, SYMBOLS or NOSYMBOLS.
   - In fill-in fields, the FIELD line should not include the word PROTECT.
   - If a fill-in field is only used when a toggle field has a specific value, then the fill-in field can be invisible most of the time.
   - Prompts, that pertain to the field that the cursor is on, appear in the prompt window (small window at the bottom of the screen).
   - Some relevant window identification numbers are

Title window	101
Menu bar	102
Tool setup window	(user specifies number)
Prompt window	3

B.2.2.4 Create SLCSE$UI:CE_WIN.BIN. Create a binary file by following the steps given below.

1. At DCL prompt, type: winnie
2. Type: O (i.e., the letter 'O', not zero)
3. Type: SLCSE$UI:CE_WIN.ASC

4. Type: Q
5. Select "SAVE BINARY"
6. Type: SLCSESUI:CE_WIN.BIN

B.2.3 Define MOO Commands

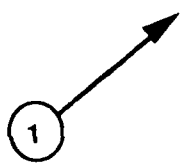
1. Determine when fields will be visible in the setup window. This is a matter of the tool integrator's preference; all of the fields can be visible at all times, or some fields may be invisible until they are required. Two questions that should be considered are
   - Is the usage of some fields dependent on the contents of other fields to generate the correct command? For example, in the BBoard setup window (figure 4-2), Field 5 (the fill-in field for the expiration date) is only required if Field 4 is toggled to the value EXPIRE; when Field 4 is toggled to the value NOEXPIRE, Field 5 is unnecessary.
   - If one field is visible, should another field be invisible? Does making one field visible, require another field to be visible also?
2. Edit SLCSESUI:CE_MOO.ASC. (Figure B-4 contains the MOO commands written for the BBoard setup window. It will be used as an example throughout the instructions for this step.)

NOTE: The window and field numbers used in the MOO commands are those defined in SLCSESUI:CE_WIN.ASC during step B.2.2. For example, the WINNIE commands for the BBoard setup window defined its window number as 324, and that is the window number used for it throughout this step.

In figure B-4, the shaded area marked 1 shows the entries made in SLCSESUI.CE_MOO.ASC for the instructions given in steps A through E. (Only the portions of Window 20's MOO commands that deal with the BBoard are shown, since the complete commands are very long.)

a. Move the cursor to the commands for Window 20. These commands consist of two main case statements. An entry for the new tool, Tool_call, must be added to each case statement. If the user presses return and all required parameters have been specified, then the selected tool gets executed. If any required parameters are not specified, then the tool setup window is displayed and the tool is not executed. If the user presses Gold key return, then the tool setup window is displayed. These commands are executed when the setup window is displayed. These commands clear the current window from the screen and begin drawing the setup window corresponding to the tool the user highlighted.

```
! MOO file for Command Executive Windows
...
! Window 20 is Tools Window for all roles
IF WINDOW = 20 AND FIELD = 0 THEN UPON . . . CRET=STAY,
CODE PARSE_INVOKE,
CASE OF (TEXT),
...
 CASE ("BBOARD"),
  CASE OF (CHECK 2),
  CASE (1), INV 99, VIS 101, ADV 102, ADV CHECK 1,
 END CASE,
...
STATUS(8) = STAY, CODE PARSE_ONLY,
CASE OF (TEXT),
...
 CASE ("BBOARD"),  INV 99, VIS 101 CHECK 1, ADV 102, FIELD 2,
...
END CASE.
...
! Window 200 is the Command (KEYWORD) Mode window.
! Status (22) means User has used Up arrow
! Status (23) means User has used Down arrow
! Status (44) means User has used Gold Up arrow
IF WINDOW = 200 AND FIELD = 1 THEN UPON STATUS(22) = CODE
RECALL_PREVIOUS;
STATUS(23) = CODE RECALL_NEXT;
STATUS(44) = CODE RECALL_ALL, GOTO 199;
CRET= CODE PARSE_INVOKE,
CASE OF (TEXT),
...
 CASE ("BBOARD"),
  CASE OF (CHECK 2),
  CASE (1),  INV 99, VIS 101, ADV 102, ADV CHECK 1,
 END CASE,
...
END CASE;
STATUS(8) = CODE PARSE_ONLY,
CASE OF (TEXT),
...
 CASE ("BBOARD"),  INV 99, VIS 101 CHECK 1, ADV 102, FIELD 2,
...
END CASE.
...
```



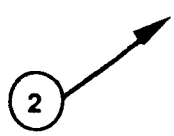


Figure B-4. MOO commands for BBoard.

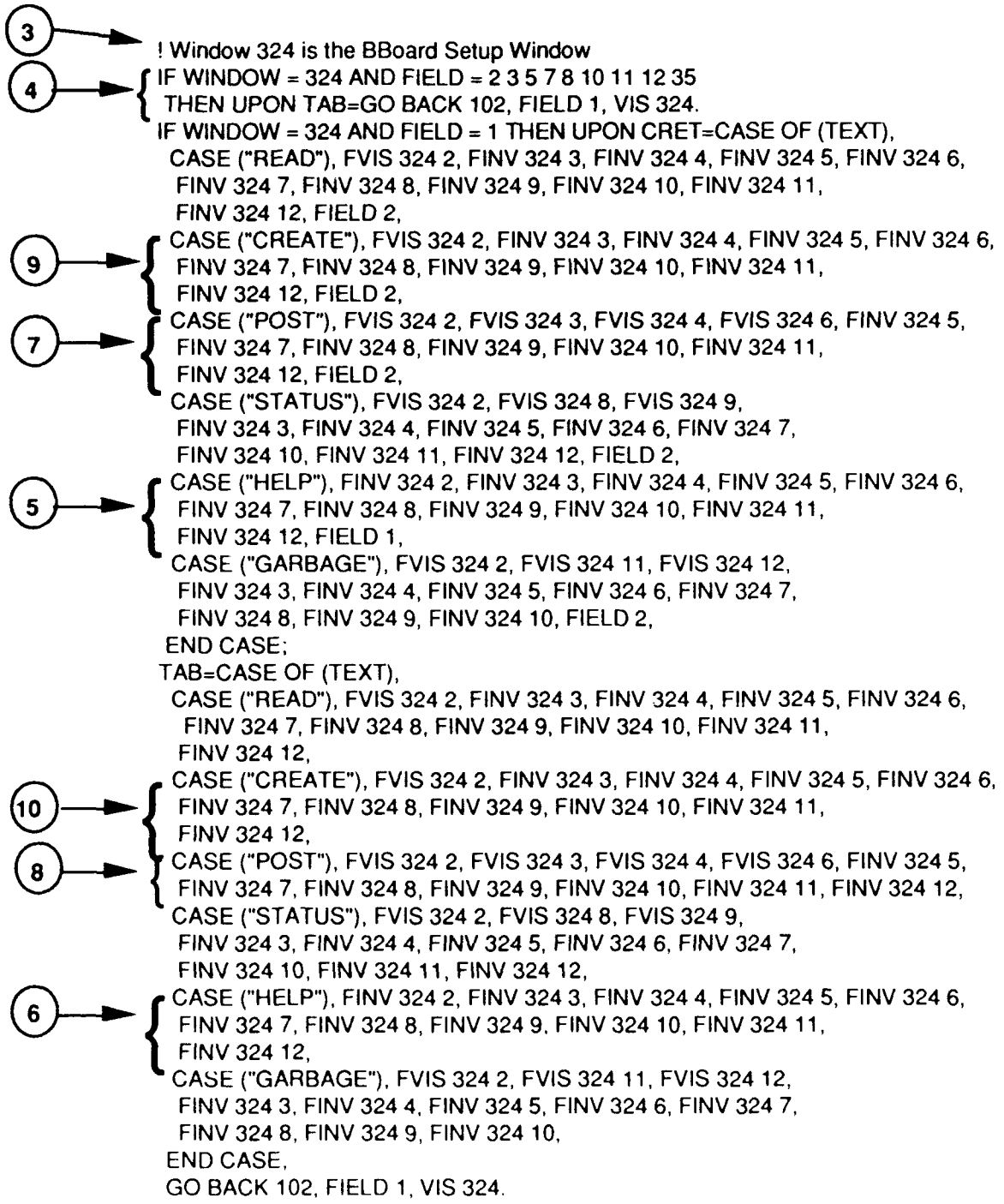


Figure B-4. MOO commands for BBoard (continued).

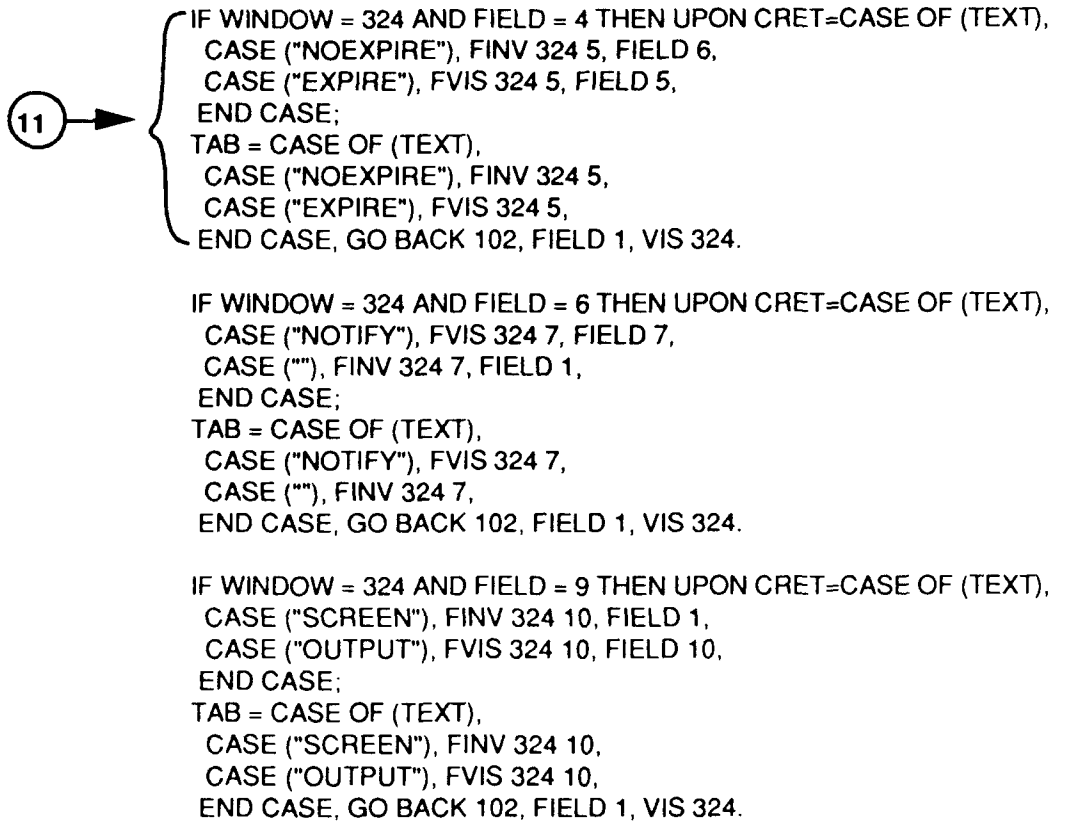


Figure B-4. MOO commands for BBOARD (continued).

b. Position the cursor on the line immediately after the statement, "IF WINDOW = 20 ... CASE OF (TEXT),".

c. Add the lines:
CASE ("TOOL_CALL"),
CASE OF (CHECK 2),
CASE (1), INV 99, VIS 101, ADV 102, ADV CHECK 1,
END CASE;

d. Position the cursor on the line immediately after the statement "(STATUS 8) ... CASE OF (TEXT),". (Status 8 indicates that the user pressed Gold key return.)

e. Add the line:
CASE ("TOOL_CALL"), INV 99, VIS 101 CHECK 1, ADV 102, FIELD 2.

In figure B-4, the shaded area marked 2 shows the entries made in SLCSESUI.CE_MOO.ASC for the instructions given in steps F through J. (Only the portions of Window 200's MOO commands that deal with the BBoard are shown, since the complete commands are very long.)

f. Move to the commands for Window 200. These commands also consist of two main case statements. These commands assure the correct response if the SLCSE is being used in keyword mode. Entries need to be made for Tool_Call, even if the tool can not be interactively executed in keyword mode, since all tools integrated into the SLCSE can be initially invoked in keyword mode.

g. Position the cursor on the line immediately after the set of statements that begin, "IF WINDOW = 200 ..." and ends with "CASE OF (TEXT),".

h. Add the lines:
```
CASE ("TOOL_CALL"),
    CASE OF (CHECK 2),
        CASE (1), INV 99, VIS 101, ADV 102, ADV CHECK 1,
END CASE;
```

i. Position the cursor on the line immediately after the statement "(STATUS 8) ... CASE OF (TEXT),".

j. Add the line:
CASE ("TOOL_CALL"), INV 99, VIS 101 CHECK 1, ADV 102, FIELD 2

k. Create the specific MOO commands for the new setup window.

NOTE: A few frequently used MOO commands are explained here. If more detailed explanations are required refer to Lamb (1989). (Throughout these explanations, x and y are integers.)

- VIS x - makes window x visible.
- INV x - makes window x invisible.
- FVIS x y - makes field y in window x visible.
- FINV x y - makes field y in window x invisible.
- CRET - is a carriage return, i.e., the statement "UPON CRET..." means upon the user entering a carriage return, do whatever follows.
- CODE name - returns *name* to the calling program, which takes the appropriate action based on the value of name.
- FIELD x - moves the cursor to field x.
- TEXT - reads the text under the cursor.

- GO BACK x - returns cursor to window x.
- RETURN - returns the cursor to the previous window

a. Position cursor where new window's instructions are to be added. The commands appear in window number order.

b. Add a comment, explaining which tool these commands correspond to. Comments are delimited with a "!". The comment for the BBoard is "! Window 324 is the BBoard Setup Window." (See 3 in figure B-4 continued.)

c. Are there any fields in the setup window that do not affect the visibility or ordering of other fields? All fill-in fields and some toggleable fields will fall into this category. These fields will follow the default action when a carriage return is entered (advancing to the next visible field), therefore, no statement is required. When the user enters a Tab or Keypad 0 the cursor will move to the menu bar. (In the SLCSE setup windows the TAB and Keypad 0 are considered identical. Entering either causes the actions defined for the TAB to be taken.) A statement is required to cause the cursor to go back to window 102 Field 1 (the INVOKE field of the setup window's menu bar). An if-then statement will be required for this action. All of these fields can be listed in one statement.

   EXAMPLE: In the BBoard setup window, Fields 2, 3, 5, 7, 8, 10, 11, 12, and 35 have no effect on the visibility of other fields in the window. Field 2, as an example, is the entry field for the name of the bulletin board. Therefore, if the screen is showing Window 324 and the cursor is in any of those fields and the user presses TAB, the cursor goes back to Window 102 Field 1. (See 4 in figure B-4.)

d. If Field 500 was used in the setup window, then the statement below must be entered for the window. (Field 500 is a standard field in the SLCSE that can be toggled between "Update with Selected Object" and "Don't Update with Selected Object".) This field serves as a flag that indicates whether or not to use the object the user has selected from the Object Menu of the SLCSE, and then position the cursor in Field 1 (INVOKE) of the setup window's menu bar. BBoard does not include Field 500, but if it did, the code in figure B-4 would need to include the statement below. The string "MODIFY_USE_OBJECT" is passed back to the calling program, where action is taken to update the file entry field (usually Field 1) based on the setting of either "UPDATE..." or "DON'T...."

   IF WINDOW = 324 AND FIELD 500 THEN UPON TAB = CODE MODIFY_USE_OBJECT, GO BACK 102, FIELD 1, VIS 324;
   CRET = CODE MODIFY_USE_OBJECT.

Notice that the window the cursor is in (324 in this example) is specifically left visible (by the command VIS 324). Any setup window that uses Field 500 will require this statement (with the correct window number) in its MOO commands.

e. If a field is toggleable and its value affects the visibility or ordering of another field in the setup window, then a statement must be created to assure the correct action is taken. It will be an if-then statement, with the "then" portion consisting of two case statements. One case statement will give commands for actions to take if the user enters a carriage return. The other case statement gives commands for actions taken if a TAB is entered.

The if-then statement will have the following syntax.

```
IF WINDOW = __ AND FIELD = __ THEN UPON CRET =
 CASE OF (TEXT),
 CASE ("value") FVIS x y, ... FINV x z, FIELD __,
 CASE...FIELD __,
...
 END CASE;
TAB = CASE OF (TEXT),
 CASE ("value") FVIS x y, ... FINV x z,
 CASE...
...
 END CASE,
GO BACK 102, FIELD 1, VIS __.
```

If the cursor is in the correct window and field and the user enters a carriage return, then TEXT reads the value contained in the field and executes the corresponding instructions from the case statement. The instructions used with a carriage return always include a specific explanation specifying where to position the cursor. Similarly, if the user enters a TAB, the function TEXT evaluates the fields value and the corresponding instruction is executed. In the case of a TAB, however, the cursor always returns to Field 1 of the menu bar and leaves setup window visible. With the exception of the FIELD instruction in the carriage return case statement's instructions, the instructions for a given entry in both case statements are identical. For example, the instructions given for Field 1 of the BBoard, value HELP are identical except for the addition of the instruction "FIELD 1." (See 5 and 6 in figure B-4.)

Each value the field can take, must have an entry in the case statement. The instruction for each possible value, defines the visibility of all the fields affected by the contents of the field. For example, Field 1 of the BBoard

window affects all fields except Fields 1 and 35. When writing the instruction, assume all the affected fields are in the wrong state; i.e., visible when they should be invisible, or vice versa.

EXAMPLE: The BBoard has six main possible actions, (1) READ from a bulletin board, (2) CREATE a bulletin board, (3) POST a message to a bulletin board, (4) find the STATUS of a bulletin board, (5) provide HELP on the BBoard command, and (6) do GARBAGE collection on a bulletin board. Field 1 of the BBoard setup window is a toggleable field with six possible values corresponding to those actions. Each action has its own required set of parameters and qualifiers. Which of the window's other fields are visible in the window, depends on the value of Field 1. For example, HELP takes no qualifiers or parameters and, therefore, has no additional fields. CREATE requires one parameter, the name of the bulletin board being created. So, the value CREATE in Field 1 requires one additional visible field. POST requires the name of the bulletin board and the name of the file containing the message to be posted. It also has two qualifiers, EXPIRE/NOEXPIRE and NOTIFY. So, if Field 1 has the value POST four additional fields need to become visible. (The first three screens shown in figure 4-2 are the setup window, with POST, CREATE, and HELP in Field 1.)

When Field 1 is HELP, Fields 2 through 12 are invisible. When a carriage return is entered, the cursor stays on Field 1. (See figure B-4, numbers 5 and 6 for the actual instructions included for Field 1, value HELP.)

When Field 1 is CREATE, Field 2 is visible and Fields 3 through 12 are invisible. When a carriage return is entered, the cursor advances to Field 2. (See figure B-4, numbers 7 and 8 for the actual instructions included for Field 1, value CREATE.)

When Field 1 is POST, Fields 2, 3, 4, and 6 are visible; while 5 and 7 through 12 are invisible. When a return is entered while in Field 1 with the value POST, the cursor is advanced to Field 2. (The resulting statements are marked 9 and 10 in figure B-4.)

EXAMPLE: In the BBoard setup window, if Field 4 is toggled to EXPIRE, then Field 5 (the expiration date) must be made visible, and the cursor advances to Field 5. If Field 4 is NOEXPIRE, then Field 5 is invisible and the cursor advances to Field 6. So, a statement must be included in the MOO commands to arrange the correct action. Field 4 affects the visibility of Field 5 only, so that is the only field included in the instruction list. (The command is number 11 in figure B-4.)

3. Create SLCSESUI:CE_MOO.BIN.

   - At prompt, type: @MOO_DIR:MOO_BIN SLCSESUI:CE_MOO.ASC

4. Copy SLCSESUI:CE_WIN.BIN to personal project directories.

   - Each person who can work in the SLCSE, has a directory for each project he/she is assigned to. For each directory type: COPY SLCSESUI:CE_WIN.BIN disk_name:[person_dir.project_dir]

B.2.4 Write Ada Procedure

Develop an Ada procedure to create a string which contains the command to run the tool. For example, if the tool is the ADAVAX compiler, the string being created might be "ADAVAX/SOURCE filename." Figure B-5 contains the INVOKE_BBOARD.ADA procedure which will be used as an example throughout this step.

1. Type: set def BASE_ROOT:[TOOL.SITE_SPECIFIC]
2. Type: copy INVOKE_SAMPLE_TOOL.ADA INVOKE_tool_call.ADA
3. Edit INVOKE_tool_call.ADA

The basic required steps are described below.

NOTES:

- During execution of this procedure the current window is the setup window for tool_call.
- A package, WINNIE, is provided. WINNIE provides a procedure named READ, which reads a specified field in the current window. The field is specified by the field number defined in step B.2.2.
- A package called TOOL_SUPPORT provides a procedure, PARSE_FILE-NAME, that checks to see if the filename the user has entered is legal and if it exists in the user's current SLCSE work space.
- An exception has already been defined for use if the specified file does not exist.
- WINDOW_ID is a pointer to the current window, i.e., the setup window. COMMAND is the string being produced, the one that invokes the tool. COMMAND_LEN is the length of COMMAND. BATCH_JOB lets the SLCSE know whether the tool is to be run in batch or interactive mode. FILES_MISSING is 0 if no parameters are missing, otherwise it contains the field number where the required parameter should have been specified.

```
-- ============================================================================
-- >>>>>>>>>>>>>>>>>>>>>> ADA COMPILATION UNIT <<<<<<<<<<<<<<<<<<<<<<<<
-- ----------------------------------------------------------------------------
-
-- NAME:        INVOKE_BBOARD
-
-- OVERVIEW:    This procedure reads the WINNIE setup window for the bboard
-              tool.  The appropriate DCL command is built; this reflects
-              the parameters and qualifiers as specified in the WINNIE
-              setup window.
-
-- FIELDS:      Field 1 - Action to be taken
-              Field 2 - Name of bulletin board
-              Field 3 - Name of file containing message
-              Field 4 - Is message to be posted with an expiration date?
-              Field 5 - Expiration date
-              Field 6 - Notify users that message has been posted?
-              Field 7 - List of users to be notified.
-              Field 8 - Full status of bulletin board?
-              Field 9 - Output status to screen or file?
-              Field 10 - Name of output file
-              Field 11 - Keep log of messages deleted during garbage
-                    collection?
-              Field 12 - Purge additional files while doing garbage
-                    collection?
-
-- RAISES:     If no bulletin board is specified, an exception is
-              raised and control returns to the calling procedure.
-              If no file is specified, or the the file specified does
-              not exist, an exception is raised and control returns to
-              the calling procedure.
-              If message expiration is specified, and no expiration
-              date is given, an exception is raised and control returns
-              to the calling procedure.
-              If status is to be output to a file and no file name is
-              specified, an exception is raised and control returns to
-              the calling procedure.
-
-- CALLS:      Called by SITE_SPECIFIC.GET_COMMAND
-
-              Calls WINNIE.read,
-              TOOL_SUPPORT.parse_filename
-              MESSAGE_DISPLAY.display_in_window
-
-- PARAMETERS: WINDOW_ID_TYPE     -- Tool setup window id
-              COMMAND            -- DCL command returned
-              COMMAND_LEN        -- Length of DCL command
-              BATCH_JOB          -- True if batch invocation, else False
-              FILES_MISSING      -- 0 if all required parameters are
-                                 specified; else, field number where
-                                 required parameter should be specified.
-
-- ============================================================================

with TOOL_SUPPORT, WINNIE, MESSAGE_DISPLAY;
```

Figure B-5. Tool invocation procedure for BBoard setup window.

```
use WINNIE;

procedure INVOKE_BBOARD ( WINDOW_ID     : in out WINNIE.WINDOW_ID_TYPE;
                          COMMAND       : in out STRING;
                          COMMAND_LEN   : in out NATURAL;
                          BATCH_JOB     : in out BOOLEAN;
                          FILES_MISSING : in out NATURAL ) is

  TEXT                : STRING (1..255);   -- Text read from WINNIE.READ
  TEXT_LENGTH         : INTEGER;           -- Length of text read from WINNIE.READ
  FILE_NAME           : STRING (1..255);   -- Parsed filename input
  FILE_LENGTH         : INTEGER;           -- Length of parsed filename
  FILE_FOUND          : BOOLEAN;           -- Whether file input exists
  ACTION              : STRING (1..255);   -- Name of main action to be taken on board
  ACTION_LENGTH       : INTEGER;           -- Length of action name
  BOARD               : STRING (1..255);   -- Name of bulletin board
  BOARD_LENGTH        : INTEGER;           -- Length of bulletin board
  TEMP_LENGTH         : INTEGER;           -- Temporary counter of command length
  NO_FILE_FOUND       : exception;         -- File not specified or non-existant
  NO_BOARD_FOUND      : exception;         -- Bulletin board not specified
  NO_DATE_FOUND       : exception;         -- Expiration date not specified
  NO_OUT_FILE_FOUND   : exception;         -- Output file not specified

begin

-- Assume all files required are specified
  FILES_MISSING := 0;

-- Start assigning DCL command string and length
  COMMAND_LEN := 6;                -- (length of tool string)
  COMMAND ( 1..COMMAND_LEN ) := "BBOARD";
  TEMP_LENGTH := COMMAND_LEN;

-- Obtain initial bboard command
  WINNIE.READ ( FIELD          => 1,
                IN_WINDOW      => WINDOW_ID,
                PUT_TEXT_IN    => ACTION,
                LENGTH_IN      => ACTION_LENGTH );

  if ACTION (1..4) = "HELP" then

    COMMAND_LEN := COMMAND_LEN + 5;
    COMMAND (TEMP_LENGTH+1..COMMAND_LEN) := "/HELP";
    TEMP_LENGTH := COMMAND_LEN;

  else

-- Get the board name
    WINNIE.READ ( FIELD          => 2,
                  IN_WINDOW      => WINDOW_ID,
                  PUT_TEXT_IN    => BOARD,
                  LENGTH_IN      => BOARD_LENGTH );

-- If no board is specified, display message and return to calling procedure
    if BOARD_LENGTH = 0 then
```

Figure B-5. Tool invocation procedure for BBoard setup window (continued).

```
      raise NO_BOARD_FOUND;
    end if;

    BATCH_JOB := False;

-- If text is READ then the default of BBOARD is invoked
    if ACTION ( 1..4 ) = "READ" then
      COMMAND_LEN := COMMAND_LEN + 1 + BOARD_LENGTH;
      COMMAND (TEMP_LENGTH+1..COMMAND_LEN) := " " & BOARD (1..BOARD_LENGTH);
      TEMP_LENGTH := COMMAND_LEN;

    else
      COMMAND_LEN := COMMAND_LEN + 1 + ACTION_LENGTH;
      COMMAND (TEMP_LENGTH+1..COMMAND_LEN) := "/" & ACTION
(1..ACTION_LENGTH);
      TEMP_LENGTH := COMMAND_LEN;

    end if;

-- Post a Message to the bulletin board?
    if ACTION (1..4) = "POST" then

-- Obtain name of file containing message to be posted
      WINNIE.READ ( FIELD          => 3,
                    IN_WINDOW      => WINDOW_ID,
                    PUT_TEXT_IN    => FILE_NAME,
                    LENGTH_IN      => FILE_LENGTH );

      if FILE_LENGTH = 0 then
        raise NO_FILE_FOUND;
      end if;

      WINNIE.READ ( FIELD          => 4,
                    IN_WINDOW      => WINDOW_ID,
                    PUT_TEXT_IN    => TEXT,
                    LENGTH_IN      => TEXT_LENGTH );

-- If EXPIRE select then find out the date of message expiration
      if TEXT (1..6) = "EXPIRE" then

        WINNIE.READ ( FIELD          => 5,
                      IN_WINDOW      => WINDOW_ID,
                      PUT_TEXT_IN    => TEXT,
                      LENGTH_IN      => TEXT_LENGTH );

        if TEXT_LENGTH = 0 then
          raise NO_DATE_FOUND;
        end if;

        COMMAND_LEN := COMMAND_LEN + 10 + TEXT_LENGTH;
        COMMAND (TEMP_LENGTH+1..COMMAND_LEN) := "/EXPIRE=""" &
        TEXT (1..TEXT_LENGTH) & """;
        TEMP_LENGTH := COMMAND_LEN;

      end if;
```

Figure B-5. Tool invocation procedure for BBoard setup window (continued).

```
      WINNIE.READ ( FIELD          => 6,
                    IN_WINDOW      => WINDOW_ID,
                    PUT_TEXT_IN    => TEXT,
                    LENGTH_IN      => TEXT_LENGTH );

-- Notify users that message is posted?
      if TEXT (1..6) = "NOTIFY" then

        COMMAND_LEN := COMMAND_LEN + 7;
        COMMAND (TEMP_LENGTH+1..COMMAND_LEN) := "/NOTIFY";
        TEMP_LENGTH := COMMAND_LEN;

        WINNIE.READ ( FIELD          => 7,
                      IN_WINDOW      => WINDOW_ID,
                      PUT_TEXT_IN    => TEXT,
                      LENGTH_IN      => TEXT_LENGTH );

        if TEXT_LENGTH /= 0 then
          COMMAND_LEN := COMMAND_LEN + 1 + TEXT_LENGTH;
          COMMAND (TEMP_LENGTH+1..COMMAND_LEN) := "=" & TEXT (1..TEXT_LENGTH);
          TEMP_LENGTH := COMMAND_LEN;

        end if;

      end if;

      COMMAND_LEN := COMMAND_LEN + 2 + FILE_LENGTH + BOARD_LENGTH;
      COMMAND (TEMP_LENGTH+1..COMMAND_LEN) := " " & BOARD (1..BOARD_LENGTH)
& " " &
        FILE_NAME (1..FILE_LENGTH);
      TEMP_LENGTH := COMMAND_LEN;

-- Create a new bulletin board?
    elsif ACTION (1..6) = "CREATE" then

      COMMAND_LEN := COMMAND_LEN + 1 + BOARD_LENGTH;
      COMMAND (TEMP_LENGTH+1..COMMAND_LEN) := " " & BOARD (1..BOARD_LENGTH);
      TEMP_LENGTH := COMMAND_LEN;

-- Obtain status of bulletin board?
    elsif ACTION (1..6) = "STATUS" then

-- Full status on bulletin board?
      WINNIE.READ ( FIELD          => 8,
                    IN_WINDOW      => WINDOW_ID,
                    PUT_TEXT_IN    => TEXT,
                    LENGTH_IN      => TEXT_LENGTH );

      if TEXT (1..4) = "FULL" then

        COMMAND_LEN := COMMAND_LEN + 5;
        COMMAND (TEMP_LENGTH+1..COMMAND_LEN) := "/FULL";
        TEMP_LENGTH := COMMAND_LEN;
```

Figure B-5. Tool invocation procedure for BBoard setup window (continued).

```
      end if;

-- Output information to file?
      WINNIE.READ ( FIELD          => 9,
                    IN_WINDOW      => WINDOW_ID,
                    PUT_TEXT_IN    => TEXT,
                    LENGTH_IN      => TEXT_LENGTH );

      if TEXT (1..6) = "OUTPUT" then

      WINNIE.READ ( FIELD          => 10,
                    IN_WINDOW      => WINDOW_ID,
                    PUT_TEXT_IN    => TEXT,
                    LENGTH_IN      => TEXT_LENGTH );

      if TEXT_LENGTH = 0 then
        raise NO_OUT_FILE_FOUND;
      end if;

      COMMAND_LEN := COMMAND_LEN + 8 + TEXT_LENGTH;
      COMMAND (TEMP_LENGTH+1..COMMAND_LEN) := "/OUTPUT=" & TEXT
(1..TEXT_LENGTH);
      TEMP_LENGTH := COMMAND_LEN;

    end if;

    COMMAND_LEN := COMMAND_LEN + 1 + BOARD_LENGTH;
    COMMAND (TEMP_LENGTH+1..COMMAND_LEN) := " " & BOARD (1..BOARD_LENGTH);
    TEMP_LENGTH := COMMAND_LEN;

-- Do Garbage Collection on bulletin board.
    elsif ACTION (1..7) = "GARBAGE" then

      WINNIE.READ ( FIELD          => 11,
                    IN_WINDOW      => WINDOW_ID,
                    PUT_TEXT_IN    => TEXT,
                    LENGTH_IN      => TEXT_LENGTH );

      if TEXT (1..4) = "LOG" then

        COMMAND_LEN := COMMAND_LEN + 4;
        COMMAND (TEMP_LENGTH+1..COMMAND_LEN) := "/LOG";
        TEMP_LENGTH := COMMAND_LEN;

      end if;

      WINNIE.READ ( FIELD          => 12,
                    IN_WINDOW      => WINDOW_ID,
                    PUT_TEXT_IN    => TEXT,
                    ENGTH_IN       => TEXT_LENGTH );

      if TEXT (1..5) = "PURGE" then

        COMMAND_LEN := COMMAND_LEN + 6;
        COMMAND (TEMP_LENGTH+1..COMMAND_LEN) := "/PURGE";
```

Figure B-5. Tool invocation procedure for BBoard setup window (continued).

```
        TEMP_LENGTH := COMMAND_LEN;

      end if;

      COMMAND_LEN := COMMAND_LEN + 1 + BOARD_LENGTH;
      COMMAND (TEMP_LENGTH+1..COMMAND_LEN) := " " & BOARD (1..BOARD_LENGTH);
      TEMP_LENGTH := COMMAND_LEN;

    end if;

  end if;

exception

  when NO_BOARD_FOUND =>
    FILES_MISSING := 1;
    MESSAGE_DISPLAY.DISPLAY_IN_WINDOW
      ( MESSAGE => "A bulletin board name is required.",
      RING_BELL => True );
  when NO_DATE_FOUND =>
    FILES_MISSING := 5;
    MESSAGE_DISPLAY.DISPLAY_IN_WINDOW
      ( MESSAGE => "A date is required.",
      RING_BELL => True );
  when NO_FILE_FOUND =>
    FILES_MISSING := 3;
    MESSAGE_DISPLAY.DISPLAY_IN_WINDOW
      ( MESSAGE => "A filename is required.",
      RING_BELL => True );
  when NO_OUT_FILE_FOUND =>
    FILES_MISSING := 10;
    MESSAGE_DISPLAY.DISPLAY_IN_WINDOW
      ( MESSAGE => "A filename is required.",
      RING_BELL => True );

end INVOKE_BBOARD;
```

Figure B-5. Tool invocation procedure for BBoard setup window (continued).

A. Change the name of the procedure in the first line from INVOKE_SAMPLE_TOOL to INVOKE_TOOL_CALL.

B. Change first assignment to COMMAND and COMMAND_LEN so that they reflect the tool being integrated.

EXAMPLE: For BBoard the statements were changed from

```
COMMAND_LEN := 11;
COMMAND (1..COMMAND_LEN) := "SAMPLE_TOOL";
```

to

```
COMMAND_LEN := 6;
COMMAND (1..COMMAND_LEN) := "BBOARD";
```

C. If the tool requires a filename as input,

   a. Edit the call to WINNIE.READ to show the correct field number. Usually the call is correct as it is shown in INVOKE_SAMPLE_TOOL, but if the filename is entered in a field other than Field 1, edit this call to reflect that.

   b. Edit the call to TOOL_SUPPORT.PARSE_FILENAME to show the correct tool name.

D. If the tool can only be used interactively,

   a. Delete the lines that read Field 35 and check the mode the tool will be used in, i.e., delete from the line that starts with "WINNIE.READ (FIELD => 35," to the following "end if;".

   b. Replace with the lines

```
BATCH_JOB := False;
MESSAGE_DISPLAY.DISPLAY_IN_WINDOW
(MESSAGE => "The tool_call has been invoked.");
```

E. If the tool can only be used in batch mode,

   a. Delete the lines that read Field 35 and check the mode the tool will be used in, i.e., delete from the line that starts with "WINNIE.READ (FIELD => 35," to the following "end if;".)

   b. Replace with the lines

```
BATCH_JOB := True;
MESSAGE_DISPLAY.DISPLAY_IN_WINDOW
(MESSAGE => "Tool_call has been sent to the batch queue.");
```

F. If all the remaining fields are toggle fields, and all the possible field values are actual values of qualifiers, i.e., "DEBUG" and "NODEB" rather than "Set Debug Option" and "Don't Set Debug," then

   a. Delete the code from present position (after batch/interactive handling code) to the line before the exception handler.

   b. Add lines that are similar to the ones below. This loop reads through the remaining fields and adds the qualifiers to the command.

```
for I in First_Field .. Last_Field loop
```

```
WINNIE.READ (FIELD        => I,
                IN_WINDOW      => WINDOW_ID,
                PUT_TEXT_IN    => TEXT,
                LENGTH_IN      => TEXT_LENGTH);
        COMMAND_LEN := COMMAND_LEN + TEXT_LENGTH + 1;
        COMMAND (TEMP_LENGTH + 1 .. COMMAND_LENGTH) :=
                "/" & TEXT (1 .. TEXT_LENGTH);
```

end loop;

G. If a remaining field is a toggle field, and the possible values do not correspond to actual qualifier values, then the field will require a statement similar to the body of the loop statement shown in F.a. However, before the correct qualifier can be added to the command some interpretation will be required.

EXAMPLE: If Field 3 has the possible values "Set Debug Option" and "Don't Set Debug" corresponding to the qualifiers "DEBUG" and "NODEB," then the resulting code would be

```
WINNIE.READ (FIELD        => 3,
        IN_WINDOW         => WINDOW_ID,
        PUT_TEXT_IN       => TEXT,
        LENGTH_IN         => TEXT_LENGTH);
if TEXT (1..3) := "Set" then
        COMMAND_LEN := COMMAND_LEN + 6;
        COMMAND (TEMP_LENGTH + 1 .. COMMAND_LENGTH) :=
                "/DEBUG";
else
        COMMAND_LEN := COMMAND_LEN + 6;
        COMMAND (TEMP_LENGTH + 1 .. COMMAND_LENGTH) :=
                "/NODEB";
end if;
```

H. If a remaining field is a fill-in field, then its contents are a parameter that has to be added to the command.

a. If the field contents are to be the name of a file that already exist, then the field can be added by copying the commands edited during step C, and editing them to reflect the correct field number.

b If the field contents are not the name of an already existing file, then something similar to the lines below must be added to the code (with, of course, the correct field number).

```
WINNIE.READ (FIELD        => 3,
                IN_WINDOW       => WINDOW_ID,
                PUT_TEXT_IN     => TEXT,
                LENGTH_IN       => TEXT_LENGTH);
COMMAND_LEN := COMMAND_LEN + TEXT_LENGTH + 1;
COMMAND (TEMP_LENGTH + 1 .. COMMAND_LENGTH) :=
      " " & TEXT (1 .. TEXT_LENGTH);
```

I. If any fill-in fields are used only upon a toggle field having a specific entry. A command similar to the one shown in the example below would be required.

EXAMPLE: BBoard Field 5 (the expiration date) only needs to be checked for a value if Field 4 has the value EXPIRE. The resulting code could be

```
WINNIE.READ (FIELD        => 4,
      IN_WINDOW       => WINDOW_ID,
      PUT_TEXT_IN     => TEXT,
      LENGTH_IN       => TEXT_LENGTH);

COMMAND_LEN := COMMAND_LEN + 1 + TEXT_LENGTH;
COMMAND (TEMP_LENGTH + 1 .. COMMAND_LEN) := "/" &
      TEXT (1..TEXT_LENGTH);
TEMP_LENGTH := COMMAND_LEN;

if TEXT(1..6) = "EXPIRE" then
      WINNIE.READ (FIELD        => 5,
                IN_WINDOW       => WINDOW_ID,
                PUT_TEXT_IN     => TEXT,
                LENGTH _IN      => TEXT_LENGTH);

if TEXT_LENGTH = 0 then
                raise NO_DATE_FOUND;
      end if;

COMMAND_LEN := COMMAND_LEN + 3 + TEXT_LENGTH;
      COMMAND (TEMP_LENGTH + 1 .. COMMAND_LENGTH) :=
                "="""" & TEXT (1..TEXT_LENGTH) & "";
      TEMP_LENGTH := COMMAND_LEN;

end if;
```

NOTE: Steps F through I can, of course, be done in other ways. For example, in step I, the code could be written so the qualifer is added only if Field 4 does not show the default value, i.e., NOEXPIRE. See figure B-5.

```
-- ================================================================================
-- >>>>>>>>>>>>>>>>>>>>>>> ADA COMPILATION UNIT <<<<<<<<<<<<<<<<<<<<<<<<
-- --------------------------------------------------------------------------------
-
-- NAME:         Procedure SITE_SPECIFIC.GET_COMMMAND
-
-- OVERVIEW:     This procedure calls procedures which build the DCL
-               command for site-specific tool invocations. The procedures
-               that are called interpret the tool setup window and build
-               the corresponding DCL command. These procedures apply
-               to tools that are integrated with a conformant user
-               interface. A call must be made in this procedure to each
-               site-specific tool invocation procedure.
-
-- PARAMETERS:  TOOL_NAME        -- Name of the tool to be invoked
-               WINDOW_ID_TYPE   -- Tool setup window id
-               COMMAND          -- DCL command returned
-               COMMAND_LEN      -- Length of DCL command
-               BATCH_JOB        -- True if batch invocation, else False
-               FILES_MISSING    -- 0 if all required parameters are
-                                specified; else, field number where
-                                required parameter should be specified.
-
-- SYSTEM:      DEC VMS Operating System
-
-- AUTHOR:      Martha Hogan
-
-- DATE:        December 8, 1989
-
-        CHANGE HISTORY
-
-- MM-DD-YY   | Initials   | Description
-- ----------------------------------------------------------------
-- 03/25/91      MLH       Added ALS/N commands
-- 05/23/91      SP        Added Bboard command
--------------------------------------------------------------------

with WINNIE;

-- Add a statement to 'with' your new setup procuedure
with INVOKE_ADAVAX, INVOKE_IMPVAX, INVOKE_LNKVAX, INVOKE_EXPVMS;
with INVOKE_BBOARD, INVOKE_LEXGEN;

separate ( SITE_SPECIFIC )

procedure GET_COMMAND ( TOOL_NAME    : in out STRING;
                        WINDOW_ID    : in out WINNIE.WINDOW_ID_TYPE;
                        COMMAND      : in out STRING;
                        COMMAND_LEN  : in out INTEGER;
                        BATCH_JOB    : in out BOOLEAN;
                        FILES_MISSING: in out INTEGER ) is
begin

  if TOOL_NAME (1..6) = "ADAVAX" then
    INVOKE_ADAVAX ( WINDOW_ID          => WINDOW_ID,
```

Figure B-6. SLCSE calling procedure for site-specific tools.

```
                    COMMAND          => COMMAND,
                    COMMAND_LEN      => COMMAND_LEN,
                    BATCH_JOB        => BATCH_JOB,
                    FILES_MISSING    => FILES_MISSING );

  elsif TOOL_NAME (1..6) = "IMPVAX" then
    INVOKE_IMPVAX ( WINDOW_ID        => WINDOW_ID,
                    COMMAND          => COMMAND,
                    COMMAND_LEN      => COMMAND_LEN,
                    BATCH_JOB        => BATCH_JOB,
                    FILES_MISSING    => FILES_MISSING );

  elsif TOOL_NAME (1..6) = "LNKVAX" then
    INVOKE_LNKVAX ( WINDOW_ID        => WINDOW_ID,
                    COMMAND          => COMMAND,
                    COMMAND_LEN      => COMMAND_LEN,
                    BATCH_JOB        => BATCH_JOB,
                    FILES_MISSING    => FILES_MISSING );

  elsif TOOL_NAME (1..6) = "EXPVMS" then
    INVOKE_EXPVMS ( WINDOW_ID        => WINDOW_ID,
                    COMMAND          => COMMAND,
                    COMMAND_LEN      => COMMAND_LEN,
                    BATCH_JOB        => BATCH_JOB,
                    FILES_MISSING    => FILES_MISSING );

  elsif TOOL_NAME (1..6) = "BBOARD" then
    INVOKE_BBOARD ( WINDOW_ID        => WINDOW_ID,
                    COMMAND          => COMMAND,
                    COMMAND_LEN      => COMMAND_LEN,
                    BATCH_JOB        => BATCH_JOB,
                    FILES_MISSING    => FILES_MISSING );

  elsif TOOL_NAME (1..6) = "LEXGEN" then
    INVOKE_LEXGEN ( WINDOW_ID        => WINDOW_ID,
                    COMMAND          => COMMAND,
                    COMMAND_LEN      => COMMAND_LEN,
                    BATCH_JOB        => BATCH_JOB,
                    FILES_MISSING    => FILES_MISSING );
  end if;

end GET_COMMAND;
```

Figure B-6. SLCSE calling procedure for site-specific tools (continued).

J. Delete all lines remaining in the file that are not used in the INVOKE_TOOL_CALL procedure but are left over from the INVOKE_SAMPLE_CALL procedure.

B.2.5 Modify And Compile Get_Command

1. Edit SITE_SPECIFIC__GET_COMMAND.ADA. Figure B-6 is this procedure as it currently exists at NOSC.

A. Add to the with statements: with INVOKE_tool_call;

B. At the end of the procedure, before the "end if," add:

```
elsif TOOL_NAME (1..9) = "TOOL_CALL" then
        INVOKE_tool_call ( WINDOW_ID      =>   WINDOW_ID,
            COMMAND                       =>   COMMAND,
            COMMAND_LEN                   =>   COMMAND_LEN,
            BATCH_JOB                     =>   BATCH_JOB,
            FILES_MISSING                 =>   FILES_MISSING );
```

NOTE: TOOL_NAME is case sensitive and must be in capital letters.

B.2.6 Relink SLCSE

1. At prompt, type: ACS SET LIB BASE_ROOT:[LIB.ADALIB]
2. Type: ADA INVOKE_tool_call
3. Type: ADA SITE_SPECIFIC__GET_COMMAND
4. SET DEF SLCSESUI
5. @BASE_ROOT:[TOOL.SITE_SPECIFIC]LINKCE
6. At prompt, type:
   INSTALL REPLACE/SHARED SLCSESUI:CE_DRIVER.EXE

The SLCSE Environment Manager is used in steps B.2.7 through B.2.10.

B.2.7 Enter Tool Name In SEM

This is accomplished by following the steps listed below.

1. At prompt, type: sem
2. Highlight selection "1. Modify the Existing Environment," <return>. (See figure A-4)
3. Return on environment name
4. Highlight "TOOLS" in menu bar, <return>. (See figure A-5)
5. On blank line type: tool_call. (See figure B-1)

NOTE: Tool_call in this step should match the one defined as the tool symbol in step B.2.1.

6. Press PF1 and <return> This will bring up the tool definition form.

B.2.8 Define Tool Parameters

The tool parameters are defined by completing the tool invocation data form shown in figure B-7. Below is a list of the fields in this form and an explanation of how to complete each. The numbers in figure B-7 correspond with the numbered explanations below.

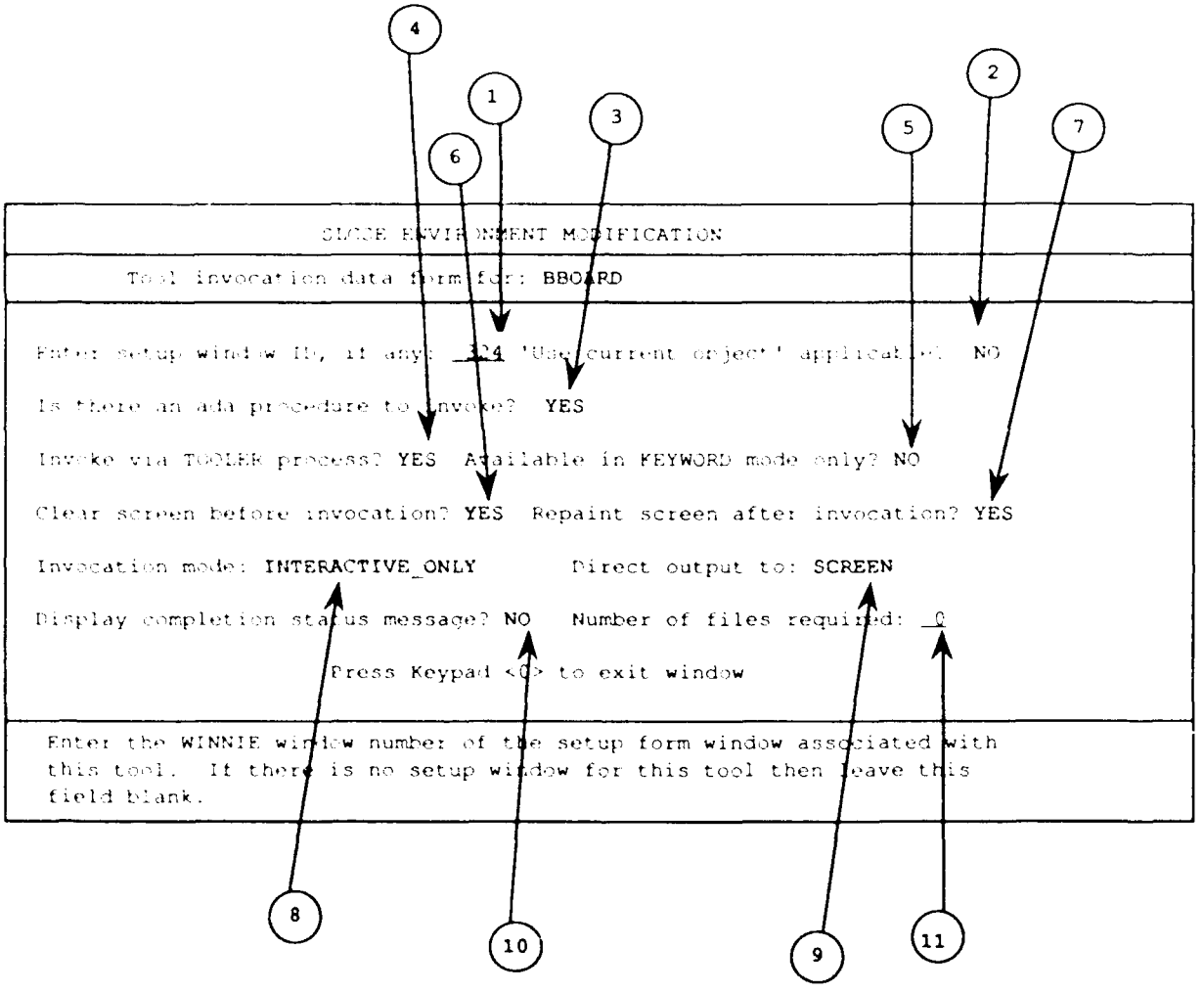


Figure B-7. Tool invocation data form for tool with qualifiers.

1. Enter Window number. <return>
2. If the tool can use a file name selected from the Object Menu toggle to "YES," otherwise toggle to "NO." <return>
3. Field should be "YES," toggle Keypad 1 if necessary. <return>
4. Field should be "YES," toggle if necessary. <return>

5. Field should be "NO," toggle if necessary. <return>
6. If tool will produce output to screen, toggle to "YES," otherwise toggle to "NO." <return>
7. If last field was "YES," then toggle to "YES," otherwise toggle to "NO." <return>
8. If tool will require user interaction before running, toggle to "INTERACTIVE_ONLY." If tool will always be run as a batch job, toggle to "BATCH_ONLY." Otherwise, toggle to "BATCH_OR_INTERACTIVE." <return>
9. If it's possible to use the tool interactively, toggle to "SCREEN," otherwise toggle to "OUTPUT_FILE." <return>
10. If it's possible to use the tool interactively, toggle to "NO," otherwise toggle to "YES." <return>
11. Enter minimum number of file names required by tool. <return>
12. Type: Keypad 0.

The form shown in figure B-2 is completed for a simple tool that produces output to the screen and can only be used interactively.

B.2.9 Associate Tool With Role(s)

Refer to figure A-5 when performing the steps given below:

1. Highlight "ROLES" in the menu bar. <return>
2. Arrow to the role that will use Tool_call, so that it is highlighted. <return>
3. Highlight Tool_call in the default tools list <return>
4. Type: Keypad 0.
5. If more than one role will use Tool_call, repeat the previous three actions.
6. Type: Keypad 0.
7. Highlight "DONE." <return>
8. <return> in response to both questions.

B.2.10 Modify Projects

1. Highlight "3. Modify existing projects." <return> (See figure A-4)
2. Highlight project to be modified. <return> and <return>

3. Modify personnel by doing the actions listed below..

   A. Highlight "PERSONNEL" in the menu bar. <return>

   B. Highlight person who will use Tool_call. Type: PF1 and <return>

   C. Highlight role the person will be using Tool_call in. Type: PF1 and <return>

   D. Verify that Tool_call is in reverse video. If it's not, highlight it. <return>

   E. Type: Keypad 0 twice.

   F. If more than one person on project will be using the tool, repeat actions B through E.

   G. Type: Keypad 0.

   H. Highlight "DONE." <return>

   I. Type "Y" in response to both questions.

4. Highlight "5. Exit the Environment Manager." <return>

APPENDIX C: SOURCE FILES FOR ALS/N WINDOWS

```
! Window 320 is the ALS/N ADAVAX Compiler Setup Window

WINDOW 320  1 80 21  5
  PRECEDENCE 111
  FRAME VIDEO "PLAIN"
  SCROLL BAR ON INSIDE RIGHT
  KEYSET    6
  TEXT     5    5 "Listing Control Options"
  TEXT    16    5 "Special Processing Options"
  TEXT    26    5 "Special Compilation Unit Options"
  FORM     1
    PROMPT   3  1   " Press the Keypad One key to toggle between options, use arrow keys
to"
    PROMPT   3  2   " navigate, or press Keypad 0 to return to the command bar."

    FIELD   35   1   5 11    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "INTERACTIVE"    1
      DEFAULT    "BATCH"          2
      LINK UPON DOWN TO FIELD   1

    FIELD  500   1 43 33    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "UPDATE WITH SELECTED OBJECT"          1
      DEFAULT    "DON'T UPDATE WITH SELECTED OBJECT"   2
      LINK UPON DOWN TO FIELD   1
      JUSTIFY RIGHT
    FIELD    1   3 33 40    "UNDERLINE" "REVERSE UNDERLINE BOLD"
      LINK UPON UP TO FIELD  500
      LABEL 3   5    "Filename to Compile" "PLAIN" "BOLD"
      PROMPT   3  1   " Use arrow keys to navigate, press Keypad 0 to return to command
bar,"
      PROMPT   3  2   " or enter ALS/N filename followed by <Return>."
      SELECT FIRST
    FIELD    2   7 49 12    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "NO_ATTRIBUTE" 1
      DEFAULT    "ATTRIBUTE"      2
      LABEL 7   5    "Produce Symbol Attribute Listing" "PLAIN" "BOLD"
    FIELD    3   8 49 14    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "NO_DIAGNOSTICS"   1
      DEFAULT    "DIAGNOSTICS"      2
      LABEL 8   5    "Produce Diagnostic Summary Listing" "PLAIN" "BOLD"
    FIELD    4   9 49 15    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "NO_MACHINE_CODE"  1
      DEFAULT    "MACHINE_CODE"     2
      LABEL 9   5    "Produce Machine Code Listing" "PLAIN" "BOLD"
    FIELD    5  10 49  8    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "NO_NOTES"  1
      DEFAULT    "NOTES"     2
      LABEL10   5    "Include Diagnostics of Note Severity" "PLAIN" "BOLD"
    FIELD    6  11 49  9    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "NO_SOURCE"    1
      DEFAULT    "SOURCE"       2
      LABEL11   5    "Produce Ada Source Listing" "PLAIN" "BOLD"
    FIELD    7  12 49 10    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
```

Figure C-1. ADAVAX setup window definitions using WINNIE.

```
  DEFAULT    "NO_SUMMARY"  1
  DEFAULT    "SUMMARY"     2
  LABEL12  5    "Produce Summary Diagnostics Listing" "PLAIN" "BOLD"
FIELD    8 13 49 18    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
  DEFAULT    "NO_CROSS_REFERENCE"    1
  DEFAULT    "CROSS_REFERENCE"       2
  LABEL13  5    "Produce Cross-Reference Listing" "PLAIN" "BOLD"
FIELD    9 14 49 10    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
  DEFAULT    "PRIVATE"       1
  DEFAULT    "NO_PRIVATE"    2
  LABEL 14 5    "Include Private Specs in Listing" "PLAIN" "BOLD"
FIELD   10 18 49  9    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
  DEFAULT    "CHECKS"        1
  DEFAULT    "NO_CHECKS"     2
  LABEL18  5    "Provide Run-time Error Checking" "PLAIN" "BOLD"
FIELD   11 19 49 18    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
  DEFAULT    "CODE_ON_WARNING"       1
  DEFAULT    "NO_CODE_ON_WARNING"    2
  LABEL19  5    "Generate Code if Warning Diagnostics" "PLAIN" "BOLD"
FIELD   12 20 49 23    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
  DEFAULT    "CONTAINER_GENERATION"       1
  DEFAULT    "NO_CONTAINER_GENERATION"    2
  LABEL  20  5    "Produce Container if Severity Permits" "PLAIN" "BOLD"
FIELD   13 21 49  8    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
  DEFAULT    "DEBUG"       1
  DEFAULT    "NO_DEBUG"    2
  LABEL  21  5    "Generate Debugger Symbols & Code" "PLAIN" "BOLD"
FIELD   15 22 49 10    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
  DEFAULT    "NO_MEASURE"    1
  DEFAULT    "MEASURE"       2
  LABEL  22  5    "Monitor Subprogram Execution Frequency" "PLAIN" "BOLD"
FIELD   16 23 49 11    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
  DEFAULT    "NO_OPTIMIZE"   1
  DEFAULT    "OPTIMIZE"      2
  LABEL  23  5    "Enable Global Optimization" "PLAIN" "BOLD"
FIELD   14 24 49 13    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
  DEFAULT    "TRACE_BACK"       1
  DEFAULT    "NO_TRACE_BACK"    2
  LABEL  24  5    "Provide Calling Sequence Traceback" "PLAIN" "BOLD"
FIELD   17 28 49 17    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
  DEFAULT    "NO_COMPILER_MAINT"  1
  DEFAULT    "COMPILER_MAINT"     2
  LABEL  28  5    "Activate All Compiler Options Below" "PLAIN" "BOLD"
FIELD   18 29 49 14    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
  DEFAULT    "NO_BIS_COMPILE"    1
  DEFAULT    "BIS_COMPILE"       2
  LABEL  29  5    "Compile Generic Built-in Subprograms" "PLAIN" "BOLD"
FIELD   19 30 49 14    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
  DEFAULT    "NO_RSL_COMPILE"    1
  DEFAULT    "RSL_COMPILE"       2
  LABEL  30  5    "Compile New ADA_RSL Package Spec" "PLAIN" "BOLD"
FIELD   20 31 49 19    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
  DEFAULT    "NO_STANDARD_COMPILE"    1
  DEFAULT    "STANDARD_COMPILE"       2
  LABEL  31  5    "Compile New STANDARD Package" "PLAIN" "BOLD"
```

Figure C-1. ADAVAX setup window definitions using WINNIE (continued).

```
FIELD   21 32 49 17    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
  DEFAULT    "NO_SYSTEM_COMPILE" 1
  DEFAULT    "SYSTEM_COMPILE"      2
  LABEL  32  5    "Compile New SYSTEM Package" "PLAIN" "BOLD"
```

Figure C-1. ADAVAX setup window definitions using WINNIE (continued).

```
! Window 322 is the ALS/N LNKVAX Setup Window

WINDOW 322  1 80 21  5
  PRECEDENCE 114
  FRAME VIDEO "PLAIN"
  SCROLL BAR ON INSIDE RIGHT
  KEYSET    6
  TEXT  10  5 "Special Processing Options"
  TEXT  17  5 "Maintenance Options"
  FORM    1
    PROMPT   3   1   " Press the Keypad One key to toggle between options, use arrow keys
to"
    PROMPT   3   2   " navigate, or press Keypad 0 to return to the command bar."

    FIELD   35   1   5 11    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "INTERACTIVE"    1
      DEFAULT    "BATCH"          2
      LINK UPON DOWN TO FIELD   1
    FIELD  500   1  43 33    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "DON'T UPDATE WITH SELECTED OBJECT"   1
      DEFAULT    "UPDATE WITH SELECTED OBJECT"         2
      INVISIBLE
      JUSTIFY RIGHT
    FIELD    1   3 33 40    "UNDERLINE" "REVERSE UNDERLINE BOLD"
      LINK UPON UP TO FIELD  35
      LABEL      3   5     "Main Subprogram" "PLAIN" "BOLD"
      PROMPT     3   1     " Enter the name of the main subprogram or the character string
'NULL'. "
      PROMPT     3   2     " 'NULL' indicates there is no main subprogram. "
      SELECT FIRST
    FIELD    2   4 33 40    "UNDERLINE" "REVERSE UNDERLINE BOLD"
      LABEL      4   5     "Output Container" "PLAIN" "BOLD"
      PROMPT     3   1     " Enter the name of container to be created by the ALS/N Linker and
placed "
      PROMPT     3   2     " in the current Program Library."
    FIELD    3   5 33 40    "UNDERLINE" "REVERSE UNDERLINE BOLD"
      LABEL      5   5     "Unit List Filename" "PLAIN" "BOLD"
      PROMPT     3   1     " Enter the file listing the Containers to be used as input for the link."
      PROMPT     3   2     " This parameter must be supplied when the main subprogram is
NULL."
    FIELD    4   6 49  8    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "NO_UNITS"    1
      DEFAULT    "UNITS"       2
      LABEL      6   5     "Produce Unit Listing" "PLAIN" "BOLD"
      PROMPT     3   1     " Press Keypad One to toggle.  UNITS indicates that a unit listing will
be"
      PROMPT     3   2     " produced."
    FIELD    5   7 49 10    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "NO_SYMBOLS"    1
      DEFAULT    "SYMBOLS"       2
      LABEL      7   5     "Produce Symbol Listing" "PLAIN" "BOLD"
      PROMPT     3   1     " Press Keypad One to toggle.  SYMBOLS indicates that a symbol
listing will "
      PROMPT     3   2     " be produced."
    FIELD    6   8 49 12    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
```

Figure C-2. LINKVAX setup window definitions using WINNIE.

```
      DEFAULT    "NO_ELAB_LIST" 1
      DEFAULT    "ELAB_LIST"      2
      LABEL      8   5     "Produce Elaboration Order Listing" "PLAIN" "BOLD"
      PROMPT     3   1     " Press Keypad One to toggle. ELAB_LIST indicates that an
elaboration "
      PROMPT     3   2     " order listing will be produced."
    FIELD      7 12 49   8     "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "NO_DEBUG"     1
      DEFAULT    "DEBUG"        2
      LABEL     12   5     "Produce Container for Debugging" "PLAIN" "BOLD"
      PROMPT     3   1     " Press Keypad One to toggle. DEBUG generates a linked container
for"
      PROMPT     3   2     " debugging. "
    FIELD      8 13 49 10     "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "NO_MEASURE"   1
      DEFAULT    "MEASURE"      2
      LABEL     13   5     "Produce Container for Performance Measure" "PLAIN" "BOLD"
      PROMPT     3   1     " Press Keypad One to toggle. MEASURE generatesa linked
container for"
      PROMPT     3   2     " performance measurement."
    FIELD      9 14 49 10     "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "NO_PARTIAL"     1
      DEFAULT    "PARTIAL"        2
      LABEL     14   5     "Permit Partial Container Creation" "PLAIN" "BOLD"
      PROMPT     3   1     " Press Keypad One to toggle. PARTIAL permits an incomplete
Container to be"
      PROMPT     3   2     " produced."
    FIELD     10 15 499 "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "SEARCH"         1
      DEFAULT    "NO_SEARCH"      2
      LABEL     15   5     "Link All Referenced Units" "PLAIN" "BOLD"
      PROMPT     3   1     " Press Keypad One to toggle. SEARCH causes all referenced units
to be"
      PROMPT     3   2     " linked; NO_SEARCH limits input Containers and routines
referenced by them."
    FIELD     11 19 49   3     "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "NO"   1
      DEFAULT    "YES"  2
      LABEL     19   5     "Propagate Linker Stack Dumps" "PLAIN" "BOLD"
    FIELD     12 20 49   3     "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "NO"   1
      DEFAULT    "YES"  2
      LABEL     20   5     "Produce Functional Trace of Execution" "PLAIN" "BOLD"
    FIELD     13 21 49   3     "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "NO"   1
      DEFAULT    "YES"  2
      LABEL     21   5     "Produce Trace of Data Transactions" "PLAIN" "BOLD"
```

Figure C-2. LINKVAX setup window definitions using WINNIE (continued).

```
WINDOW 323  1 80 21  5
  PRECEDENCE 115
  FRAME VIDEO "PLAIN"
  SCROLL BAR ON INSIDE RIGHT
  KEYSET    6
  TEXT      10  5    "Special Processing Options"
  TEXT      17  5    "Maintenance Options"
  FORM    1
    PROMPT 3   1    " Press the Keypad One key to toggle between options, use arrow keys
to"
    PROMPT 3   2    " navigate, or press Keypad 0 to return to the command bar."

    FIELD   35   1   5  11    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "INTERACTIVE"    1
      DEFAULT    "BATCH"          2
      LINK UPON DOWN TO FIELD   1
    FIELD  500   1  43  33    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "DON'T UPDATE WITH SELECTED OBJECT"   1
      DEFAULT    "UPDATE WITH SELECTED OBJECT"         2
      INVISIBLE
      JUSTIFY RIGHT
    FIELD    1   3  33  40    "UNDERLINE" "REVERSE UNDERLINE BOLD"
      LINK UPON UP TO FIELD  35
      LABEL    3   5    "Linked Container" "PLAIN" "BOLD"
      PROMPT 3   1    " Enter the name of the linked Container for the program that is to be"
      PROMPT 3   2    " exported."
      SELECT FIRST
    FIELD    2   4  33  40    "UNDERLINE" "REVERSE UNDERLINE BOLD"
      LABEL    4   5    "Export Module" "PLAIN" "BOLD"
      PROMPT 3   1    " Enter filename where the executable load module is to be stored."
    FIELD    3   5  33  40    "UNDERLINE" "REVERSE UNDERLINE BOLD"
      LABEL    5   5    "Directive File" "PLAIN" "BOLD"
      PROMPT 3   1    " Enter the filename where exporter directives are contained."
    FIELD    4   7  49   6    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "NO_MAP"        1
      DEFAULT    "MAP"           2
      LABEL    7   5    "Produce Program Sections Map Listing" "PLAIN" "BOLD"
      PROMPT 3   1    " Press Keypad One to toggle.  MAP indicates that a program sections
map"
      PROMPT 3   2    " summarizing the executable image will be produced."
    FIELD    5   8  49  10    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "NO_SYMBOLS"   1
      DEFAULT    "SYMBOLS"      2
      LABEL    8   5    "Produce Symbol Listing" "PLAIN" "BOLD"
      PROMPT 3   1    " Press Keypad One to toggle.  SYMBOLS indicates that a symbol
listing will "
      PROMPT 3   2    " be produced; NO_SYMBOLS indicates that a symbol listing won't be
produced."
    FIELD    6  12  49  13    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "NO_ACCOUNTING"   1
      DEFAULT    "ACCOUNTING"      2
      LABEL   12   5    "Report Elapsed CPU and Wall Clock Time" "PLAIN" "BOLD"
      PROMPT 3   1    " Press Keypad One to toggle.  ACCOUNTING reports the elapsed
CPU and wall"
      PROMPT 3   2    " clock time at program termination in the message output file."
```

Figure C-3. EXPVMS setup window definitions using WINNIE.

```
    FIELD     7 13 49  8     "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "NO_DEBUG"    1
      DEFAULT    "DEBUG"       2
      LABEL  13  5     "Allow Use of Symbolic Debugger" "PLAIN" "BOLD"
      PROMPT 3   1     " Press Keypad One to toggle. DEBUG activates the Debugger Kernel
in the"
      PROMPT 3   2     " program image to allow use of an Ada Program Symbolic Debugger."
    FIELD     8 14 49 10     "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "NO_MEASURE"  1
      DEFAULT    "MEASURE"     2
      LABEL  14  5     "Perform Frequency Analysis" "PLAIN" "BOLD"
      PROMPT 3   1     " Press Keypad One to toggle. MEASURE activates the Frequency
and "
      PROMPT 3   2     " Statistical Analyzer Kernel so frequency analysis can be performed."
    FIELD     9 15 49 16     "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "NO_DEBUG_SYMBOLS" 1
      DEFAULT    "DEBUG_SYMBOLS"    2
      LABEL  15  5     "Produce Symbols List for Debugger" "PLAIN" "BOLD"
      PROMPT 3   1     " Press Keypad One to toggle. DEBUG causes the Exporter to
produce a list of"
      PROMPT 3   2     " external symbols suitable for use by the VAX/VMS Debugger."
    FIELD   10 19 49  3     "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "NO"          1
      DEFAULT    "YES"         2
      LABEL  19  5     "Propagate Exporter Stack Dumps" "PLAIN" "BOLD"
    FIELD   11 20 49  3     "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "NO"          1
      DEFAULT    "YES"         2
      LABEL  20  5     "Produce Functional Trace of Execution" "PLAIN""BOLD"
    FIELD   12 21 49  3     "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "NO"          1
      DEFAULT    "YES"         2
      LABEL  21  5     "Produce Trace of Data Transactions" "PLAIN" "BOLD"
```

Figure C-3. EXPVMS setup window definitions using WINNIE (continued).

```
! Window 321 is the ALS/N IMPVAX Setup Window

WINDOW 321  1 80 21  5
  PRECEDENCE 113
  FRAME VIDEO "PLAIN"
  SCROLL BAR ON INSIDE RIGHT
  KEYSET    6
  TEXT  11   5 "Maintenance Options"
  FORM    1
    PROMPT 3   1     " Press the Keypad One key to toggle between options use arrow keys
to"
    PROMPT 3   2     " navigate, or press Keypad 0 to return to the command bar."

    FIELD   35   1   5 11    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "INTERACTIVE"    1
      DEFAULT    "BATCH"          2
      LINK UPON DOWN TO FIELD   1
    FIELD  500   1  43 33    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "DON'T UPDATE WITH SELECTED OBJECT"   1
      DEFAULT    "UPDATE WITH SELECTED OBJECT"         2
      INVISIBLE
      JUSTIFY RIGHT
    FIELD    1   3 33 40    "UNDERLINE" "REVERSE UNDERLINE BOLD"
      LINK UPON UP TO FIELD  35
      LABEL 3   5    "Import Module" "PLAIN" "BOLD"
      PROMPT   3    1        " Enter the name of the host file that stores the program object
module to "
      PROMPT   3    2        " import."
      SELECT FIRST
    FIELD    2   5 33 40    "UNDERLINE" "REVERSE UNDERLINE BOLD"
      LABEL 5   5     "Output Container" "PLAIN" "BOLD"
      PROMPT   3    1        " Enter the name of the unit body to be created in the
designated program"
      PROMPT   3    2        " library. This unit is either a package body or
    FIELD    3   7 33 40    "UNDERLINE" "REVERSE UNDERLINE BOLD"
      LABEL 7   5    "Directive File" "PLAIN" "BOLD"
      PROMPT   3    1        " Enter file which supplies an entry point and reference
information about"
      PROMPT   3    2        " the file being imported. This option is required for package
bodies."
    FIELD    4   9 33 10    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "NO_PACKAGE"   1
      DEFAULT    "PACKAGE"      2
      LABEL 9   5    "Unit is a Package Body" "PLAIN" "BOLD"
      PROMPT   3    1        " Press Keypad One to toggle. PACKAGE indicates that unit is a
package "
      PROMPT   3    2        " body and directive file is required; NO_PACKAGE indicates
subprogram body."
    FIELD    5 13 49   3    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "NO"   1
      DEFAULT    "YES"  2
      LABEL13   5    "Propagate Importer Stack Dumps" "PLAIN" "BOLD"
    FIELD    6 14 49   3    "PLAIN" "REVERSE BOLD" "PLAIN" PROTECT
      DEFAULT    "NO"   1
      DEFAULT    "YES"  2
```

Figure C-4. IMPVMS setup window definitions using WINNIE.

```
  LABEL14   5     "Produce Functional Trace of Execution"  "PLAIN" "BOLD"
FIELD     7 15 49   3     "PLAIN"  "REVERSE BOLD"  "PLAIN"  PROTECT
  DEFAULT     "NO"   1
  DEFAULT     "YES"  2
  LABEL15   5     "Produce Trace of Data Transactions"  "PLAIN"  "BOLD"
```

Figure C-4. IMPVMS setup window definitions using WINNIE (continued).

```
! MOO file for Command Executive Windows
. . .
! Window 20 is Tools Window for all roles
IF WINDOW = 20 AND FIELD = 0 THEN UPON . . . CRET=STAY, CODE PARSE_INVOKE,
CASE OF (TEXT),
  CASE ("ADAVAX"),
    CASE OF (CHECK 2),
    CASE (1), INV 99, VIS 101, ADV 102, ADV CHECK 1,
  END CASE,
  CASE ("IMPVAX"),
    CASE OF (CHECK 2),
    CASE (1), INV 99, VIS 101, ADV 102, ADV CHECK 1,
  END CASE,
  CASE ("LNKVAX"),
    CASE OF (CHECK 2),
    CASE (1), INV 99, VIS 101, ADV 102, ADV CHECK 1,
  END CASE,
  CASE ("EXPVMS"),
    CASE OF (CHECK 2),
    CASE (1), INV 99, VIS 101, ADV 102, ADV CHECK 1,
  END CASE,
  . . .
STATUS(8) = STAY, CODE PARSE_ONLY,
CASE OF (TEXT),
  CASE ("ADAVAX"), INV 99, VIS 101 CHECK 1, ADV 102, FIELD 2,
  CASE ("IMPVAX"), INV 99, VIS 101 CHECK 1, ADV 102, FIELD 2,
  CASE ("LNKVAX"), INV 99, VIS 101 CHECK 1, ADV 102, FIELD 2,
  CASE ("EXPVMS"), INV 99, VIS 101 CHECK 1, ADV 102, FIELD 2,
. . .
END CASE.
. . .
! Window 200 is the Command (KEYWORD) Mode window.
! Status (22) means User has used Up arrow
! Status (23) means User has used Down arrow
! Status (44) means User has used Gold Up arrow
IF WINDOW = 200 AND FIELD = 1 THEN UPON STATUS(22) = CODE RECALL_PREVIOUS;
STATUS(23) = CODE RECALL_NEXT;
STATUS(44) = CODE RECALL_ALL, GOTO 199;
CRET= CODE PARSE_INVOKE,
CASE OF (TEXT),
  CASE ("ADAVAX"),
    CASE OF (CHECK 2),
    CASE (1), INV 99, VIS 101, ADV 102, ADV CHECK 1,
  END CASE,
  CASE ("IMPVAX"),
    CASE OF (CHECK 2),
    CASE (1), INV 99, VIS 101, ADV 102, ADV CHECK 1,
  END CASE,
  CASE ("LNKVAX"),
    CASE OF (CHECK 2),
    CASE (1), INV 99, VIS 101, ADV 102, ADV CHECK 1,
  END CASE,
  CASE ("EXPVMS"),
    CASE OF (CHECK 2),
    CASE (1), INV 99, VIS 101, ADV 102, ADV CHECK 1,
```

Figure C-5. MOO commands for ALS/N tools.

```
  END CASE,
...
END CASE;
STATUS(8) = CODE PARSE_ONLY,
CASE OF (TEXT),
  CASE ("ADAVAX"),  INV 99, VIS 101 CHECK 1, ADV 102, FIELD 2,
  CASE ("IMPVAX"),  INV 99, VIS 101 CHECK 1, ADV 102, FIELD 2,
  CASE ("LNKVAX"),  INV 99, VIS 101 CHECK 1, ADV 102, FIELD 2,
  CASE ("EXPVMS"),  INV 99, VIS 101 CHECK 1, ADV 102, FIELD 2,
...
END CASE.
...
! Window 320 is the ALS/N ADAVAX Setup Window
IF WINDOW = 320 AND FIELD = 1:35 THEN UPON TAB=GO BACK 102, FIELD 1, VIS 320.
IF WINDOW = 320 AND FIELD = 500 THEN UPON TAB=CODE MODIFY_USE_OBJECT,
  GO BACK 102, FIELD 1, VIS 320; CRET=CODE MODIFY_USE_OBJECT.

! Window 321 is the ALS/N IMPVAX Setup Window
IF WINDOW = 321 AND FIELD = 1:35 THEN UPON TAB=GO BACK 102, FIELD 1, VIS 321.
IF WINDOW = 321 AND FIELD = 500 THEN UPON TAB=CODE MODIFY_USE_OBJECT,
  GO BACK 102, FIELD 1, VIS 321; CRET=CODE MODIFY_USE_OBJECT.

! Window 322 is the ALS/N LNKVAX Setup Window
IF WINDOW = 322 AND FIELD = 1:35 THEN UPON TAB=GO BACK 102, FIELD 1, VIS 322.
IF WINDOW = 322 AND FIELD = 500 THEN UPON TAB=CODE MODIFY_USE_OBJECT,
  GO BACK 102, FIELD 1, VIS 322; CRET=CODE MODIFY_USE_OBJECT.

! Window 323 is the ALS/N EXPVMS Setup Window
IF WINDOW = 323 AND FIELD = 1:35 THEN UPON TAB=GO BACK 102, FIELD 1, VIS 323.
IF WINDOW = 323 AND FIELD = 500 THEN UPON TAB=CODE MODIFY_USE_OBJECT,
  GO BACK 102, FIELD 1, VIS 323; CRET=CODE MODIFY_USE_OBJECT.
```

Figure C-5. MOO commands for ALS/N tools (continued).

```
-- ================================================================
-- >>>>>>>>>>>>>>>>>>>> ADA COMPILATION UNIT <<<<<<<<<<<<<<<<<<<<<<<
-- ----------------------------------------------------------------
-
-- NAME:        INVOKE_ADAVAX
-
-- AUTHOR:      Martha Hogan
-
-- DATE:        March 26, 1991
-
-- ABSTRACT:  This procedure interprets the WINNIE Setup Window for the
-            ALS/N Ada compiler and builds the appropriate DCL command.
-            The content of the ALS/N Ada Setup Window is listed below:
-
-       Field 1:   Filename to Compile
-       Field 2:   Produce Symbol Attribute Listing
-       Field 3:   Produce Diagnostic Summary Listing
-       Field 4:   Produce Machine Code Listing
-       Field 5:   Include Diagnostics of Note Severity
-       Field 6:   Produce Ada Source Listing
-       Field 7:   Produce Summary Diagnostics Listing
-       Field 8:   Produce Cross-Reference Listing
-       Field 9:   Include Private Specs in Listing
-
-       Field 10:  Provide Run-time Error Checking
-       Field 11:  Generate Code if Warning Diagnostics
-       Field 12:  Produce Container if Severity Permits
-       Field 13:  Generate Debugger Symbols & Code
-       Field 14:  Monitor Subprogram Execution Frequency
-       Field 15:  Enable Global Optimization
-       Field 16:  Provide Calling Sequence Traceback
-
-       Field 17:  Activate All Compiler Options Below
-       Field 18:  Compile Generic Built-in Subprograms
-       Field 19:  Compile New ADA_RSL Package Spec
-       Field 20:  Compile New STANDARD Package
-       Field 21:  Compile New SYSTEM Package
-       Field 35:  Interactive/Batch flag
-
-       CHANGE HISTORY
-
-- MM-DD-YY    | Initials | Description
-- ----------------------------------------------------------------
-- 07-22-91       MLH     Changed code so qualifiers are added to command
-                          only if not the default.
----------------------------------------------------------------------

with TOOL_SUPPORT, WINNIE, MESSAGE_DISPLAY;
use WINNIE;

procedure INVOKE_ADAVAX ( WINDOW_ID       : in WINNIE.WINDOW_ID_TYPE;
                          COMMAND         : in out STRING;
                          COMMAND_LEN     : in out NATURAL;
                          BATCH_JOB       : in out BOOLEAN;
                          FILES_MISSING   : out NATURAL ) is
```

Figure C-6. Tool invocation procedure for ADAVAX setup window.

```
  FILE_FOUND        : BOOLEAN;          -- Status returned from PARSE_FILENAME
  TEXT              : STRING (1..255);  -- Text returned from WINNIE.READ
  TEXT_LENGTH       : NATURAL;          -- Length of TEXT
  FILE_NAME         : STRING (1..255);  -- Filename returned from PARSE_FILENAME
  FILE_LENGTH       : NATURAL;          -- Length of FILE_NAME
  OLD_LENGTH        : INTEGER;          -- Length of command string

begin

-- Read Field 1 to obtain filename to compile.
  WINNIE.READ ( FIELD           => 1,
                IN_WINDOW       => WINDOW_ID,
                PUT_TEXT_IN     => TEXT,
                LENGTH_IN       => TEXT_LENGTH );

-- If the user hasn't specified a filename, display a message and return
-- to the ADAVAX setup menu.
  if TEXT_LENGTH = 0 then
    FILES_MISSING := 1;
    MESSAGE_DISPLAY.DISPLAY_IN_WINDOW
      ( MESSAGE => "A filename must be specified.", RING_BELL => True );
    return;
  end if;

-- Check syntax of filename.
  TOOL_SUPPORT.PARSE_FILENAME ( INPUT_FILE          => TEXT (1..TEXT_LENGTH),
                                DEFAULT_SPEC        => ".ADA",
                                TOOL_NAME           => "ADAVAX",
                                LOGICAL_PREFIX      => True,
                                ENTIRE_FILE_SPEC    => FILE_NAME,
                                ENTIRE_FILE_LENGTH  => FILE_LENGTH,
                                FILE_FOUND          => FILE_FOUND,
                                MULT_FILES_ALLOWED  => False );
  if not FILE_FOUND then
    FILES_MISSING := 1;
    return;
  else
    FILES_MISSING := 0;
  end if;

-- Process the rest of the command.
  COMMAND_LEN := 6;
  COMMAND ( 1..COMMAND_LEN ) := "ADAVAX";
  OLD_LENGTH := COMMAND_LEN;

-- Read Fields 2 through 8 to obtain command qualifiers.
  for I in 2..8 loop
    WINNIE.READ ( FIELD           => WINNIE.FIELD_ID_TYPE (I),
                  IN_WINDOW       => WINDOW_ID,
                  PUT_TEXT_IN     => TEXT,
                  LENGTH_IN       => TEXT_LENGTH );

    if TEXT(1..3) /= "NO_" then
      COMMAND_LEN := COMMAND_LEN + 1 + TEXT_LENGTH;
```

Figure C-6. Tool invocation procedure for ADAVAX setup window (continued).

```
    COMMAND ( OLD_LENGTH+1 .. COMMAND_LEN ) := "/" & TEXT (1..TEXT_LENGTH);
    OLD_LENGTH := COMMAND_LEN;
  end if;

end loop;

for I in 9..13 loop
  WINNIE.READ ( FIELD          => WINNIE.FIELD_ID_TYPE (I),
                IN_WINDOW      => WINDOW_ID,
                PUT_TEXT_IN    => TEXT,
                LENGTH_IN      => TEXT_LENGTH );

  if TEXT(1..3) = "NO_" then
    COMMAND_LEN := COMMAND_LEN + 1 + TEXT_LENGTH;
    COMMAND ( OLD_LENGTH+1 .. COMMAND_LEN ) := "/" & TEXT (1..TEXT_LENGTH);
    OLD_LENGTH := COMMAND_LEN;
  end if;

end loop;

for I in 14..15 loop
  WINNIE.READ ( FIELD          => WINNIE.FIELD_ID_TYPE (I),
                IN_WINDOW      => WINDOW_ID,
                PUT_TEXT_IN    => TEXT,
                LENGTH_IN      => TEXT_LENGTH );

  if TEXT(1..3) /= "NO_" then
    COMMAND_LEN := COMMAND_LEN + 1 + TEXT_LENGTH;
    COMMAND ( OLD_LENGTH+1 .. COMMAND_LEN ) := "/" & TEXT (1..TEXT_LENGTH);
    OLD_LENGTH := COMMAND_LEN;
  end if;

end loop;

WINNIE.READ ( FIELD          => 16,
              IN_WINDOW      => WINDOW_ID,
              PUT_TEXT_IN    => TEXT,
              LENGTH_IN      => TEXT_LENGTH );

if TEXT(1..3) = "NO_" then
  COMMAND_LEN := COMMAND_LEN + 1 + TEXT_LENGTH;
  COMMAND ( OLD_LENGTH+1 .. COMMAND_LEN ) := "/" & TEXT (1..TEXT_LENGTH);
  OLD_LENGTH := COMMAND_LEN;
end if;

for I in 17..21 loop
  WINNIE.READ ( FIELD          => WINNIE.FIELD_ID_TYPE (I),
                IN_WINDOW      => WINDOW_ID,
                PUT_TEXT_IN    => TEXT,
                LENGTH_IN      => TEXT_LENGTH );

  if TEXT(1..3) /= "NO_" then
    COMMAND_LEN := COMMAND_LEN + 1 + TEXT_LENGTH;
    COMMAND ( OLD_LENGTH+1 .. COMMAND_LEN ) := "/" & TEXT (1..TEXT_LENGTH);
    OLD_LENGTH := COMMAND_LEN;
```

Figure C-6. Tool invocation procedure for ADAVAX setup window (continued).

```
    end if;

  end loop;

-- Concatenate filename to end of command.
  COMMAND_LEN := COMMAND_LEN + 1 + FILE_LENGTH;
  COMMAND (OLD_LENGTH+1..COMMAND_LEN) := " " & FILE_NAME (1..FILE_LENGTH);

-- Read Field 35 to see if interactive or batch execution.
  WINNIE.READ ( FIELD          => 35,
                IN_WINDOW      => WINDOW_ID,
                PUT_TEXT_IN    => TEXT,
                LENGTH_IN      => TEXT_LENGTH );

  if TEXT ( 1..11 ) = "INTERACTIVE" then
    BATCH_JOB := False;
    MESSAGE_DISPLAY.DISPLAY_IN_WINDOW
      ( MESSAGE => "The ALS/N Ada compiler has been invoked." );
  else
    BATCH_JOB := True;
    MESSAGE_DISPLAY.DISPLAY_IN_WINDOW
      ( MESSAGE => "ADAVAX job has been sent to the batch queue." );
  end if;

end INVOKE_ADAVAX;
```

Figure C-6. Tool invocation procedure for ADAVAX setup window (continued).

```
-- ================================================================================
-- >>>>>>>>>>>>>>>>>>>>>> ADA COMPILATION UNIT <<<<<<<<<<<<<<<<<<<<<<<<
-- --------------------------------------------------------------------------------
-
-- NAME:        INVOKE_LNKVAX
-
-- AUTHOR:      Martha Hogan
-
-- DATE:        March 28, 1991
-
-- ABSTRACT:  This procedure interprets the WINNIE Setup Window for the
-            ALS/N Linker and builds the appropriate DCL command.  The
-            Content of the Linker Setup Window is listed below:
-
-       Field 1:   Main Subprogram
-       Field 2:   Output Container
-       Field 3:   Unit List Filename
-       Field 4:   Produce Unit Listing
-       Field 5:   Produce Symbol Listing
-       Field 6:   Produce Elaboration Order Listing
-       Field 7:   Produce Container for Debugging
-       Field 8:   Produce Container for Performance Measure
-       Field 9:   Permit Partial Container Creation
-       Field 10:  Link All Referenced Units
-       Field 11:  Propagate Linker Stack Dumps
-       Field 12:  Produce Functional Trace of Execution
-       Field 13:  Produce Trace of Data Transactions
-       Field 35.  Batch/Interactive Flag
-
-       CHANGE HISTORY
-
-- MM-DD-YY    | Initials | Description
-- ----------------------------------------------------------------------------
-- 07/25/91       MLH      Fixed message displayed when LNKVAX is invoked
-                           interactively.
--------------------------------------------------------------------------------

with TOOL_SUPPORT, WINNIE, MESSAGE_DISPLAY;
use WINNIE;

procedure INVOKE_LNKVAX ( WINDOW_ID          : in WINNIE.WINDOW_ID_TYPE;
                          COMMAND            : in out STRING;
                          COMMAND_LEN        : in out NATURAL;
                          BATCH_JOB          : in out BOOLEAN;
                          FILES_MISSING      : out NATURAL ) is

  FILE_FOUND             : BOOLEAN;          -- Status returned from PARSE_FILENAME
  TEXT                   : STRING (1..255);  -- Text returned from WINNIE.READ
  TEXT_LENGTH            : NATURAL;          -- Length of TEXT
  FILE_NAME              : STRING (1..255);  -- Filename returned from
PARSE_FILENAME
  FILE_LENGTH            : NATURAL;          -- Length of FILE_NAME
  OLD_LENGTH             : INTEGER;          -- Length of command string

  FILE_NOT_SPECIFIED     : exception;
```

Figure C-7. Tool invocation procedure for LNKVAX setup window.

```
begin

  FILES_MISSING := 0;

-- Start building the command.
  COMMAND_LEN := 6;
  COMMAND ( 1..COMMAND_LEN ) := "LNKVAX";
  OLD_LENGTH := COMMAND_LEN;

  for I in 1..2 loop

-- Read Fields 1 & 2 to obtain Main Program and Output Container.
    WINNIE.READ ( FIELD          => WINNIE.FIELD_ID_TYPE (I),
                  IN_WINDOW      => WINDOW_ID,
                  PUT_TEXT_IN    => TEXT,
                  LENGTH_IN      => TEXT_LENGTH );

-- If the user hasn't specified filename, set FILES_MISSING parameter
-- to the field where filename is missing, and raise an exception
-- A message will be displayed and the user will return to the
-- appropriate field in the setup window,

    if TEXT_LENGTH = 0 then
      FILES_MISSING := I;
      raise FILE_NOT_SPECIFIED;
    end if;

    COMMAND_LEN := COMMAND_LEN + 1 + TEXT_LENGTH;
    COMMAND ( OLD_LENGTH+1 .. COMMAND_LEN ) := " " & TEXT (1..TEXT_LENGTH);
    OLD_LENGTH := COMMAND_LEN;

  end loop;

-- Read Field 3 to obtain Unit List file.  Specification of Unit List file
-- is required only if main subprogram is NULL.

  WINNIE.READ ( FIELD          => 3,
                IN_WINDOW      => WINDOW_ID,
                PUT_TEXT_IN    => TEXT,
                LENGTH_IN      => TEXT_LENGTH );

  if TEXT_LENGTH /= 0 then
    COMMAND_LEN := COMMAND_LEN + 11 + TEXT_LENGTH;
    COMMAND (OLD_LENGTH+1..COMMAND_LEN) := " /UNITLIST=" & TEXT
(1..TEXT_LENGT
H);
    OLD_LENGTH := COMMAND_LEN;
  end if;

-- Read Fields 4 through 9 to obtain qualifiers.  The default for Fields
-- 4 through 9 is the negated qualifier; that is, preceded with "NO_".
-- Add qualifier to command only if not the default.

  for I in 4..9 loop
```

Figure C-7. Tool invocation procedure for LNKVAX setup window (continued).

```
    WINNIE.READ ( FIELD          => WINNIE.FIELD_ID_TYPE (I),
                  IN_WINDOW      => WINDOW_ID,
                  PUT_TEXT_IN    => TEXT,
                  LENGTH_IN      => TEXT_LENGTH );

    if TEXT (1..3) /= "NO_" then
      COMMAND_LEN := COMMAND_LEN + 1 + TEXT_LENGTH;
      COMMAND ( OLD_LENGTH+1 .. COMMAND_LEN ) := "/" & TEXT (1..TEXT_LENGTH);
      OLD_LENGTH := COMMAND_LEN;
    end if;
  end loop;

-- Read Field 10 for SEARCH/NO_SEARCH qualifier.
  WINNIE.READ ( FIELD          => 10,
                IN_WINDOW      => WINDOW_ID,
                PUT_TEXT_IN    => TEXT,
                LENGTH_IN      => TEXT_LENGTH );

  COMMAND_LEN := COMMAND_LEN + 1 + TEXT_LENGTH;
  COMMAND ( OLD_LENGTH+1 .. COMMAND_LEN ) := "/" & TEXT (1..TEXT_LENGTH);
  OLD_LENGTH := COMMAND_LEN;

-- Read Fields 11 through 13 to obtain qualifiers. The default for Fields
-- 11 through 13 is NO. Add qualifier to command only if not the default.

  for I in 11..13 loop
    WINNIE.READ ( FIELD          => WINNIE.FIELD_ID_TYPE (I),
                  IN_WINDOW      => WINDOW_ID,
                  PUT_TEXT_IN    => TEXT,
                  LENGTH_IN      => TEXT_LENGTH );

    if TEXT (1..2) /= "NO" then
      COMMAND_LEN := COMMAND_LEN + 1 + TEXT_LEN
      COMMAND (OLD_LENGTH+1..COMMAND_LEN) := "/" &  EXT (1..TEXT_LENGTH);
      OLD_LENGTH := COMMAND_LEN;
    end if;
  end loop;

-- Read Field 35 to see if interactive or batch execution.
  WINNIE.READ ( FIELD          => 35,
                IN_WINDOW      => WINDOW_ID,
                PUT_TEXT_IN    => TEXT,
                LENGTH_IN      => TEXT_LENGTH );

  if TEXT ( 1..11 ) = "INTERACTIVE" then
    BATCH_JOB := False;
    MESSAGE_DISPLAY.DISPLAY_IN_WINDOW
      ( MESSAGE => "The ALS/N Linker has been invoked." );
  else
    BATCH_JOB := True;
    MESSAGE_DISPLAY.DISPLAY_IN_WINDOW
      ( MESSAGE => "LNKVAX job has been sent to the batch queue." );
  end if;

exception
```

Figure C-7. Tool invocation procedure for LNKVAX setup window (continued).

```
  when FILE_NOT_SPECIFIED =>
    MESSAGE_DISPLAY.DISPLAY_IN_WINDOW
      ( MESSAGE   => "A filename must be specified.", RING_BELL => True );

end INVOKE_LNKVAX;
```

Figure C-7. Tool invocation procedure for LNKVAX setup window (continued).

```
-- ================================================================================
-- >>>>>>>>>>>>>>>>>>>>>>>> ADA COMPILATION UNIT <<<<<<<<<<<<<<<<<<<<<<<<
-- --------------------------------------------------------------------------------
--
-- NAME:        INVOKE_EXPVMS
--
-- AUTHOR:      Martha Hogan
--
-- DATE:        March 28, 1991
--
-- ABSTRACT:  This procedure interprets the WINNIE Setup Window for the
--            ALS/N Exporter and builds the appropriate DCL command.  The
--            content of the Exporter Setup Window is listed below:
--
--       Field 1:   Linked Container
--       Field 2:   Export Module
--       Field 3:   Directive File
--       Field 4:   Produce Program Sections Map Listing
--       Field 5:   Produce Symbol Listing
--       Field 6:   Report Elapsed CPU and Wall Clock Time
--       Field 7:   Allow Use of Symbolic Debugger
--       Field 8:   Perform Frequency Analysis
--       Field 9:   Produce Symbols List for Debugger
--       Field 10:  Propagate Exporter Stack Dumps
--       Field 11:  Produce Functional Trace of Execution
--       Field 12:  Produce Trace of Data Transactions
--       Field 35:  Batch/Interactive Flag
--
--       CHANGE HISTORY
--
-- MM-DD-YY     | Initials | Description
-- ----------------------------------------------------------------
----------------------------------------------------------------

with TOOL_SUPPORT, WINNIE, MESSAGE_DISPLAY;
use WINNIE;

procedure INVOKE_EXPVMS ( WINDOW_ID        : in WINNIE.WINDOW_ID_TYPE;
                          COMMAND          : in out STRING;
                          COMMAND_LEN      : in out NATURAL;
                          BATCH_JOB        : in out BOOLEAN;
                          FILES_MISSING    : out NATURAL ) is

  FILE_FOUND          : BOOLEAN;          -- Status returned from PARSE_FILENAME
  TEXT                : STRING (1..255);  -- Text returned from WINNIE.READ
  TEXT_LENGTH         : NATURAL;          -- Length of TEXT
  FILE_NAME           : STRING (1..255);  -- Filename returned from PARSE_FILENAME
  FILE_LENGTH         : NATURAL;          -- Length of FILE_NAME
  OLD_LENGTH          : INTEGER;          -- Length of command string

  FILE_NOT_SPECIFIED  : exception;

begin

  FILES_MISSING := 0;
```

Figure C-8. Tool invocation procedure for EXPVMS setup window.

```
-- Start building the command.
  COMMAND_LEN := 6;
  COMMAND ( 1..COMMAND_LEN ) := "EXPVMS";
  OLD_LENGTH := COMMAND_LEN;

  for I in 1..2 loop

-- Read Fields 1 & 2 to obtain Linked Container and Export Module.
  WINNIE.READ ( FIELD          => WINNIE.FIELD_ID_TYPE (I),
                IN_WINDOW      => WINDOW_ID,
                PUT_TEXT_IN    => TEXT,
                LENGTH_IN      => TEXT_LENGTH );

-- If the user hasn't specified filename, set FILES_MISSING parameter
-- to the field where filename is missing, and raise an exception.
-- A message will be displayed and the user will return to the
-- appropriate field in the setup window,

    if TEXT_LENGTH = 0 then
      FILES_MISSING := I;
      raise FILE_NOT_SPECIFIED;
    end if;

    COMMAND_LEN := COMMAND_LEN + 1 + TEXT_LENGTH;
    COMMAND ( OLD_LENGTH+1 .. COMMAND_LEN ) := " " & TEXT (1..TEXT_LENGTH);
    OLD_LENGTH := COMMAND_LEN;

  end loop;

-- Read Field 3 to obtain Directives file. Specification of the Directives
-- file is optional.

  WINNIE.READ ( FIELD          => 3,
                IN_WINDOW      => WINDOW_ID,
                PUT_TEXT_IN    => TEXT,
                LENGTH_IN      => TEXT_LENGTH );

  if TEXT_LENGTH /= 0 then
    COMMAND_LEN := COMMAND_LEN + 13 + TEXT_LENGTH;
    COMMAND ( OLD_LENGTH+1 .. COMMAND_LEN ) := " /DIRECTIVES=" & TEXT
(1..TEXT_LENGTH);
    OLD_LENGTH := COMMAND_LEN;
  end if;

-- Read Fields 4 through 9 to obtain qualifiers. The default for Fields
-- 4 through 9 is the negated qualifier; that is, preceded with "NO_".
-- Add qualifier to command only if not the default.

  for I in 4..9 loop
    WINNIE.READ ( FIELD          => WINNIE.FIELD_ID_TYPE (I),
                  IN_WINDOW      => WINDOW_ID,
                  PUT_TEXT_IN    => TEXT,
                  LENGTH_IN      => TEXT_LENGTH );
```

Figure C-8. Tool invocation procedure for EXPVMS setup window (continued).

```
    if TEXT (1..3) /= "NO_" then
      COMMAND_LEN := COMMAND_LEN + 1 + TEXT_LENGTH;
      COMMAND ( OLD_LENGTH+1 .. COMMAND_LEN ) := "/" & TEXT (1..TEXT_LENGTH);
      OLD_LENGTH := COMMAND_LEN;
    end if;
  end loop;

-- Read Fields 10 through 12 to obtain qualifiers. The default for Fields
-- 10 through 12 is NO. Add qualifier to command only if not the default.

  for I in 10..12 loop
    WINNIE.READ ( FIELD          => WINNIE.FIELD_ID_TYPE (I),
                  IN_WINDOW      => WINDOW_ID,
                  PUT_TEXT_IN    => TEXT,
                  LENGTH_IN      => TEXT_LENGTH );

    if TEXT (1..2) /= "NO" then
      COMMAND_LEN := COMMAND_LEN + TEXT_LENGTH + 1;
      COMMAND (OLD_LENGTH+1..COMMAND_LEN) := "/" & TEXT (1..TEXT_LENGTH);
      OLD_LENGTH := COMMAND_LEN;
    end if;
  end loop;

-- Read Field 35 to see if interactive or batch execution.
  WINNIE.READ ( FIELD          => 35,
                IN_WINDOW      => WINDOW_ID,
                PUT_TEXT_IN    => TEXT,
                LENGTH_IN      => TEXT_LENGTH );

  if TEXT ( 1..11 ) = "INTERACTIVE" then
    BATCH_JOB := False;
    MESSAGE_DISPLAY.DISPLAY_IN_WINDOW
      ( MESSAGE => "The ALS/N Exporter has been invoked." );
  else
    BATCH_JOB := True;
    MESSAGE_DISPLAY.DISPLAY_IN_WINDOW
      ( MESSAGE => "EXPVMS job has been sent to the batch queue." );
  end if;

exception

  when FILE_NOT_SPECIFIED =>
    MESSAGE_DISPLAY.DISPLAY_IN_WINDOW
      ( MESSAGE  => "A filename must be specified.", RING_BELL => True );

end INVOKE_EXPVMS;
```

Figure C-8. Tool invocation procedure for EXPVMS setup window (continued).

```
-- ==========================================================================
 - >>>>>>>>>>>>>>>>>>>>>> ADA COMPILATION UNIT <<<<<<<<<<<<<<<<<<<<<<<
-- --------------------------------------------------------------------------
-
-- NAME:        INVOKE_IMPVAX
-
-- AUTHOR:      Martha Hogan
-
-- DATE:        March 28, 1991
-
-- ABSTRACT:  This procedure interprets the WINNIE Setup Window for the
-             ALS/N Importer and builds the appropriate DCL command.  The
-             content of the Importer Setup Window is listed below:
-
-       Field 1:        Import Module
-       Field 2:        Output Container
-       Field 3:        Directive File
-       Field 4:        Unit is a Package Body
-       Field 5:        Propagate Importer Stack Dumps
-       Field 6:        Produce Functional Trace of Execution
-       Field 7:        Produce Trace of Data Transactions
-
-       CHANGE HISTORY
-
-- MM-DD-YY    | Initials | Description
-- ------------------------------------------------------------------
---------------------------------------------------------------------------

with TOOL_SUPPORT, WINNIE, MESSAGE_DISPLAY;
use WINNIE;

procedure INVOKE_IMPVAX ( WINDOW_ID          in WINNIE.WINDOW_ID_TYPE;
                          COMMAND            : in out STRING;
                          COMMAND_LEN        : in out NATURAL;
                          BATCH_JOB          : in out BOOLEAN;
                          FILES_MISSING      : out NATURAL ) is

  FILE_FOUND            : BOOLEAN;      -- Status returned from PARSE_FILENAME
  TEXT                  : STRING (1..255); -- Text returned from WINNIE.READ
  TEXT_LENGTH           : NATURAL;      -- Length of TEXT
  FILE_NAME             : STRING (1..255); -- Filename returned from PARSE_FILENAME
  FILE_LENGTH           : NATURAL;      -- Length of FILE_NAME
  OLD_LENGTH            : INTEGER;      -- Length of command string

  FILE_NOT_SPECIFIED    : exception;

begin

  FILES_MISSING := 0;

-- Start building the command.
  COMMAND_LEN := 6;
  COMMAND ( 1..COMMAND_LEN ) := "IMPVAX";
  OLD_LENGTH := COMMAND_LEN;
```

Figure C-9. Tool invocation procedure for IMPVAX setup window.

```
  for I in 1..2 loop

-- Read Fields 1 & 2 to obtain Import Module and Output Container
  WINNIE.READ ( FIELD          => WINNIE.FIELD_ID_TYPE (I),
                IN_WINDOW      => WINDOW_ID,
                PUT_TEXT_IN    => TEXT,
                LENGTH_IN      => TEXT_LENGTH );

-- If the user hasn't specified filename, set FILES_MISSING parameter
-- to the field where filename is missing, and raise an exception.
-- A message will be displayed and the user will return to the
-- appropriate field in the setup window,

      if TEXT_LENGTH = 0 then
      FILES_MISSING := I;
      raise FILE_NOT_SPECIFIED;
    end if;

    COMMAND_LEN := COMMAND_LEN + 1 + TEXT_LENGTH;
    COMMAND ( OLD_LENGTH+1 .. COMMAND_LEN ) := " " & TEXT (1..TEXT_LENGTH);
    OLD_LENGTH := COMMAND_LEN;

  end loop;

-- Read Field 3 to obtain Directives File.  Specification of Directives
-- File is optional.

  WINNIE.READ ( FIELD          => 3,
                IN_WINDOW      => WINDOW_ID,
                PUT_TEXT_IN    => TEXT,
                LENGTH_IN      => TEXT_LENGTH );

  if TEXT_LENGTH /= 0 then
    COMMAND_LEN := COMMAND_LEN + 1 + TEXT_LENGTH;
    COMMAND ( OLD_LENGTH+1 .. COMMAND_LEN ) := " " & TEXT (1..TEXT_LENGTH);
    OLD_LENGTH := COMMAND_LEN;
  end if;

-- Add space to command before concatenating qualifiers.
  COMMAND_LEN := COMMAND_LEN + 1;
  COMMAND ( COMMAND_LEN .. COMMAND_LEN ) := " ";
  OLD_LENGTH := COMMAND_LEN;

-- Read Fields 4 through 7 to obtain qualifiers.  The default for Field 4
-- is NO_PACKAGE; the default for Fields 5 through 7 is NO.  Add qualifier
-- to command only if not the default.

  for I in 4..7 loop
    WINNIE.READ ( FIELD          => WINNIE.FIELD_ID_TYPE (I),
                  IN_WINDOW      => WINDOW_ID,
                  PUT_TEXT_IN    => TEXT,
                  LENGTH_IN      => TEXT_LENGTH );

    if TEXT (1..2) /= "NO" then
      COMMAND_LEN := COMMAND_LEN + 1 + TEXT_LENGTH;
```

Figure C-9. Tool invocation procedure for IMPVAX setup window (continued).

```
      COMMAND (OLD_LENGTH+1..COMMAND_LEN) := "/" & TEXT(1..TEXT_LENGTH);
      OLD_LENGTH := COMMAND_LEN;

    end if;
  end loop;

-- Read Field 35 to see if interactive or batch execution.
  WINNIE.READ ( FIELD          => 35,
                IN_WINDOW      => WINDOW_ID,
                PUT_TEXT_IN    => TEXT,
                LENGTH_IN      => TEXT_LENGTH );

  if TEXT ( 1..11 ) = "INTERACTIVE" then
    BATCH_JOB := False;
    MESSAGE_DISPLAY.DISPLAY_IN_WINDOW
      ( MESSAGE => "The ALS/N Importer has been invoked." );
  else
    BATCH_JOB := True;
    MESSAGE_DISPLAY.DISPLAY_IN_WINDOW
      ( MESSAGE => "IMPVAX job has been sent to the batch queue." );
  end if;

exception

  when FILE_NOT_SPECIFIED =>
    MESSAGE_DISPLAY.DISPLAY_IN_WINDOW
      ( MESSAGE  => "A filename must be specified.", RING_BELL => True );

end INVOKE_IMPVAX;
```

Figure C-9. Tool invocation procedure for IMPVAX setup window (continued).

APPENDIX D: DESCRIPTION OF TOOLS ADDED TO SLCSE BY NOSC

LGEN

This language generator tool was developed by B. Meyers, and A. Smith of NSWC, Dahlgren. It is written in Ada and runs on VAX/VMS. The tool accepts as input a formal definition of a language in a Backus-Naur form (BNF). From the specification of the grammar, LGEN generates elements of the language. The report by Meyers (1988) contains examples of how to use LGEN to generate (1) Ada type declarations, (2) an assembler language, and (3) reading material for elementary school children. LGEN has been used extensively for the Ship Gridlock project at NSWC, Dahlgren, and was used on the Inertial Navigation System project at the SEI to generate test messages.

ADA BULLETIN BOARD

This electronic bulletin board program was developed by L. Madden, K. Schumaker, and B. Meyers of NSWC, Dahlgren. This program is written entirely in Ada and runs on VAX/VMS. This electronic bulletin board supports features common to other bulletin boards, such as, reading and posting messages, searching for character strings, and saving messages to files. Other features that are supported include (1) posting messages with expiration dates, (2) posting messages with an option to send electronic mail, (3) accessing messages on a screen basis with screen browsing support, (4) explicit deletion of messages, and (5) a garbage collection capability. A discussion of the Ada Bulletin Board user interface and system management operations is found in Madden (1989).

LEXGEN

This lexical analyzer generator tool was developed by A. Smith and B. Meyers of NSWC, Dahlgren. It is written in Ada and runs on VAX/VMS. The tool accepts as input a specification of the tokens of a language and generates Ada code that may be used to scan an input stream. Unique features of LEXGEN include (1) procedures to return tokens from either a file or a buffer, (2) capability to return multiple tokens for a given input character sequence, (3) capability to return line and column number of the token location, (4) automatic conversion of lexeme to a particular case, and (5) considerable error processing. A description of LEXGEN and examples of its use is found in Smith (1989).

APRICOT

The Ada Primitive Compilation Order Tool (APRICOT) is a portable compilation order tool that was developed by R. Ollerton of NOSC. It was written in Ada and runs of VAX/VMS (DEC ADA), Sun (Alsys), and other compilers/computers. Tool features include the following: (1) generates compilation order from either a file containing a list of file names or a concatenated source file in pager format, (2) automatic compilation command files from either the file of file names or concatenated pager file, (3) capability to page and unpage files, and others. The tool offers both a command line interface and a menu interface with a help facility. A users guide for this tool is scheduled to be written in FY 92.

BMD

The Bit-Oriented Message Definer (BMD) is a prototype tool for defining bit-oriented messages in Ada. This tool was developed by H. Mumm and S. Parker, NOSC, as an Independent Exploratory Development (IED) project. BMD is written in Ada and runs on VAX/VMS (DEC Ada), Sun (Telesoft), and Sun (Alsys) computers/compilers. The BMD defines messages using records and record representation specifications. BMD generates source code for five different target computers. This source code varies from one target computer to another because of differences in how bits are numbered, how they are ordered, and the size of type integer. A preliminary version of this tool was used by Science Applications International Corporation (SAIC) to generate approximately 7000 Ada source lines of code for the Ada Bit-Oriented Message Handler (ABOM) project. A description of the tool and examples is found in Mumm (1990).

PRETTY PRINTER

This pretty printer is Pretty Printer 6 from the Ada Software Repository (ASR) at White Sands, New Mexico. It was developed by A. Shell, AdaCraft, Incorporated, for the NASA/Goddard Space Flight Center. This tool was written in Ada and runs on VAX/VMS (DEC Ada), SUN (Verdix), PS (Alsys) and other compilers/computers. Pretty Printer 6 implements many of the directives in the Proposed MIL-HDBL-1804, "Ada Style Guide." This tool is also in the AdaNet repository. The modifications required to change the number of columns of indentation, the case of variable names, and other features of the pretty printer are isolated in one Ada package specification. This pretty printer was used by NOSC, Code 854, for the Ada discrete-event simulation research conducted for the Shared Adaptive Internetworking (SAINT) project. A machine-readable users guide for the pretty printer and Proposed MIL-HDBL-1804 are in the ASR and AdaNet.

ADA LINE COUNTER

The Ada line counter is File_Checker from which came from the ASR. This program was written by R. Conn, T.I. and Management Assistance Corporation of America (MACA). Fixes have been made by H. Mumm, NOSC, and P. Babick, SAIC. Ada line counter is written in Ada and runs on VAX/VMS (DEC Ada), Sun (Verdix), and probably all validated Ada compilers. The program counts Ada source lines of code several different ways. This tool counts statements ending with delimiting semicolons, comments, statements ending with delimiting semicolons plus comments, and "card image" statements or lines. This tool was used on the ABOM project by NOSC and SAIC to report programmer productivity statistics. This tool has also been used at NOSC on the Joint Automated Message Editing System (JAMES) project.

BODY STUBBER

The body stubber is Body Stubber 2 from the ASR. This tool was written by J. Orost, Concurrent Computer Corporation. It was upgraded by N. Tran, NOSC, so that it will run on VAX/VMS (DEC Ada) and other validated Ada compilers. This tool is useful when developing large systems in Ada where it is essential to define the interfaces very early. The body stubber reads in an Ada package specification that contains the specification for subprograms and tasks and automatically creates a compilable package body containing stubs for the subprograms and tasks.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1 AGENCY USE ONLY *(Leave blank)*	2 REPORT DATE	3 REPORT TYPE AND DATES COVERED
	November 1991	Final

4 TITLE AND SUBTITLE

EXTENSIBILITY EXPERIMENTS WITH THE SOFTWARE LIFE-CYCLE SUPPORT ENVIRONMENT

5 FUNDING NUMBERS

PR: ECB3
WU: DN088524
PE: 0602234N

6 AUTHOR(S)

S. A. Parker, R. H. Mumm

7 PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Ocean Systems Center
San Diego, CA 92152-5000

8 PERFORMING ORGANIZATION REPORT NUMBER

NOSC TR 1463

9 SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Naval Ocean Systems Center
Block Programs
San Diego, CA 92152-5000

10 SPONSORING/MONITORING AGENCY REPORT NUMBER

11 SUPPLEMENTARY NOTES

12a DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b DISTRIBUTION CODE

13 ABSTRACT *(Maximum 200 words)*

This report describes the research carried out on the Software Life-Cycle Support Environment (SLCSE) by the Software Engineering Environment (SEE) prototypes task of the Software Engineering for C^3 Systems project. The focus of this investigation is to perform extensibility experiments with the SLCSE. These experiments included the development of an interface to the ALS/N ADAVAX compiler and the integration of a number of public domain and other no-cost software tools into the SLCSE. One goal of this research was to determine if the SLCSE could be tailored to meet the needs of a specific project.

14 SUBJECT TERMS

software engineering technology
hierarchical development technology
HDM
POD

15 NUMBER OF PAGES

124

16 PRICE CODE

17 SECURITY CLASSIFICATION OF REPORT	18 SECURITY CLASSIFICATION OF THIS PAGE	19 SECURITY CLASSIFICATION OF ABSTRACT	20 LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAME AS REPORT

21a NAME OF RESPONSIBLE INDIVIDUAL	21b TELEPHONE *(Include Area Code)*	21c OFFICE SYMBOL
S. A. Parker	(619) 553-5120	Code 411

INITIAL DISTRIBUTION

Code 0012	Patent Counsel	(1)
Code 0142	K. Campbell	(1)
Code 0144	R. November	(1)
Code 40	R. C. Kolb	(1)
Code 402	R. A. Wasilausky	(1)
Code 41	A. Justice	(1)
Code 411	J. Schulte	(1)
Code 411	R. Holmes	(1)
Code 411	H. Mumm	(1)
Code 411	S. Parker	(1)
Code 411	S. Rotter	(1)
Code 411	M. Shapiro	(1)
Code 411	N. Tran	(1)
Code 961	Archive/Stock	(6)
Code 964B	Library	(3)

Defense Technical Information Center
Alexandria, VA 22304-6145 (4)

NCCOSC Washington Liaison Office
Washington, DC 20363-5100

Navy Acquisition, Research & Development Information Center (NARDIC)
Pasadena, CA 91106-3955

Information Resources Management
Washington, DC 20350-1000

Office of Naval Research
Arlington, VA 22217-5000 (3)

Office of Naval Technology
Arlington, VA 22217-5000

Naval Undersea Systems Center
Newport, RI 02841

Naval Surface Warfare Center
Dahlgren, VA 22428

Naval Command, Control & Ocean Surveillance Center
RDT&E Division Detachment
Warminster, PA 18974-5000 (?)

AMSEL-RD-SE-AST
Fort Monmouth, NJ 07703-5000

Boeing Aerospace
Seattle, WA 98124

C.S. Draper Laboratory, Inc.
Cambridge, MA 02139

STARS Technology Center
Arlington, VA 22203

Center for Naval Analyses
Alexandria, VA 22302-0268

Navy Acquisition, Research & Development Information Center (NARDIC)
Alexandria, VA 22333

ODDRE (R&AT)/SCT
Washington, DC 20301-3080

Defense Advanced Research Projects Agency
Arlington, VA 22209-2308 (2)

Space & Naval Warfare Systems Command
Washington, DC 20363-5100

Naval Weapons Center
China Lake, CA 93555-6001

Naval Research Laboratory
Washington, DC 20375-5000

Naval Surface Warfare Center
Silver Spring, MD 20390-5000

Naval Postgraduate School
Monterey, CA 93943

Rome Air Development Center/COE
Griffiss AFB, NY 13441 (2)

Carnegie-Mellon University
Pittsburgh, PA 15213

IBM Corporation
Gaithersburg, MD 20879

ISSI
Austin, TX 78759